

**Relatório Trabalho - Turma P01**

Trabalho realizado por:

Gonçalo Lopes, nº 107572

Miguel Miragaia, nº 108317

Indice

[Introdução 4](#_Toc124075009)

[Menu 5](#_Toc124075010)

[Opções implementadas 6](#_Toc124075011)

[Opção 1 6](#_Toc124075012)

[Opção 2 8](#_Toc124075013)

[Opção 3 12](#_Toc124075014)

[Opção 4 17](#_Toc124075015)

[Escolha de Valores 21](#_Toc124075016)

[Conclusão 22](#_Toc124075017)

Índice de Figuras

[Figura 1 - menu 5](file:///C:\Users\gonca\OneDrive%20-%20Universidade%20de%20Aveiro\Relatório_MPEI%20-%20Final.docx#_Toc124074613)

[Figura 2 - variáveis opção 1 file1 7](#_Toc124074614)

[Figura 3 - função getUsers() 7](#_Toc124074615)

[Figura 4 - opção 1 8](#_Toc124074616)

[Figura 5 - opção 2 9](#_Toc124074617)

[Figura 6 - variáveis opção 2 file2 10](#_Toc124074618)

[Figura 7 - função minHasFilms() 10](#_Toc124074619)

[Figura 8 - função getDistancesMinHashFilms() 10](#_Toc124074620)

[Figura 9 - variáveis da opção 2 file2 11](#_Toc124074621)

[Figura 10 - procura de 2 filmes similares 11](#_Toc124074622)

[Figura 11 - print da opção 2 12](#_Toc124074623)

[Figura 12 - getInteresses() 13](#_Toc124074624)

[Figura 13 - erro do unique() 13](#_Toc124074625)

[Figura 14 - variáveis opção 3 file1 14](#_Toc124074626)

[Figura 15 - função matrizAssinaturas() 14](#_Toc124074627)

[Figura 16 - função minHash() 15](#_Toc124074628)

[Figura 17 - função getDistancesMinHashInteresses() 16](#_Toc124074629)

[Figura 18 – opção 3 16](#_Toc124074630)

[Figura 19 - função matrizMinHashInteresses() 16](#_Toc124074631)

[Figura 20 - filtersimilarinteresses() 17](#_Toc124074632)

[Figura 21 - opção 4 18](#_Toc124074633)

[Figura 22 - variáveis opção 4 file1 18](#_Toc124074634)

[Figura 23 - minHashTitles() 19](#_Toc124074635)

[Figura 24 - função getdistancesMinHashtitles() 19](#_Toc124074636)

[Figura 25 - função searchTitles() 20](#_Toc124074637)

[Figura 26 - função filterSimilar 21](#_Toc124074638)

# Introdução

No contexto da disciplina de MPEI (Métodos Probabilísticos para Engenharia Informática), foi-nos proposta a realização de um trabalho prático sobre algoritmos probabilísticos. Temos como objetivo o desenvolvimento de uma aplicação em Matlab, com funcionalidades de um sistema online de disponibilização de filmes.

A aplicação pede ao utilizador para inserir um id de um filme para com ele podermos executar algumas das seguintes opções:

* 1 - Users that evaluated current movie
* 2 - Suggestion of users to evaluate movie
* 3 - Suggestion of users based on common intererts
* 4 – Movies feedback based on popularity

# Menu

Para a criação do menu recorremos à função *menu()* que é disponibilizada pelo *Matlab.* Esta função apresenta as opções como um botão e retorna o índice de 1 a 6 do botão selecionado ou 0 caso se clique no botão de fechar janela.

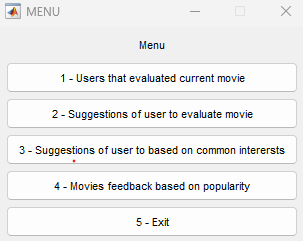


Figura 1 - menu

# Opções implementadas

## Opção 1

Na opção 1 é pedido para listar os nomes dos utilizadores que avaliaram o filme atual, primeiramente carregámos as informações dos ficheiros “u.data”, “users.txt”, “films\_info.txt” para as variáveis *udata, dic, dic2*(usando *cell array*), respetivamente.

O ficheiro “u.data”, tem na primeira coluna os *ID’s* dos utilizadores, na segunda coluna os *IDs dos filmes avaliados pelos utilizadores dessa mesma linha e na terceira coluna a avaliação feita por esse utilizador a esse filme.*

Criámos uma matriz com a primeira, segunda e terceira coluna da variável *udata*. Usando a segunda coluna desta, criámos um *array* *films* com os *ID’s* dos diferentes filmes usando a função *unique* disponibilizada pelo *matlab* que escolhe só retorna os filmes diferentes.

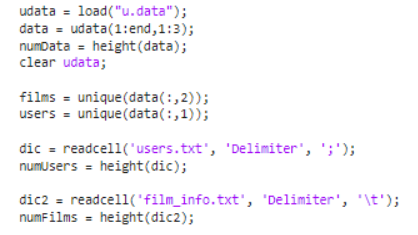


Figura 2 - variáveis opção 1 file1

De seguida criámos um *cell array* com uma coluna e com um número de linhas igual ao número de filmes diferentes. Cada linha tem um *cell array* com os *ID*’s dos utilizadores que viram o filme n. Fizemos isto através da função *getUsers().* Nesta função passamos como argumento duas variáveis(*films* e *data*), já dentro da função criamos a variável *numFilms* que guarda o número total de filmes, esta vai ser utilizada no ciclo *for* para percorrer os *ID’s de* todos os filmes e guardar na variável *indices* os índices do array *films* e depois acrescentamos ao *array usersE* o *ID* do filme *n* e o *ID* do utilizador que avaliou esse filme.

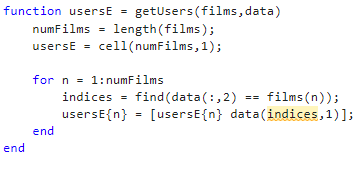


Figura 3 - função getUsers()

No fim já temos todos os utilizadores que avaliaram o filme, logo basta usar o *ID* do filme para ir buscar os users que estão no array useresE. Por último fizemos um ciclo *for* para percorrer todos os utilizadores e usamos as variaveis *idUser, nameUser, apelidoUser* para dar o print desejado.

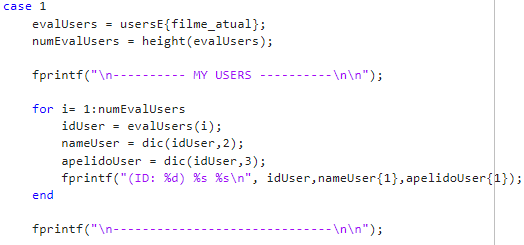


Figura 4 - opção 1

## Opção 2

Na opção 2, é pedido para determinarmos os 2 filmes mais similares ao filme atual, em termos de conjunto de utilizadores que avaliaram cada filme, e listar os *IDs* e os nomes dos utilizadores que avaliaram pelo menos um dos filmes selecionados, mas não avaliaram o atual.

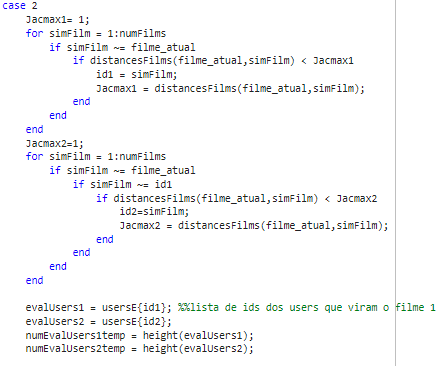


Figura 5 - opção 2

No início desta opção inicializamos a variável *Jacmax1, que limita a distância máxima de Jaccard, com valor de 1, usamos um ciclo for* para a variável *SimFilm (Filme Similar)* que percorre desde o valor 1 até ao valor de *numFilms,* variável que contem o número total de Filmes. Através da condição *if* verificamos se este é diferente do *id* filme atual, para que não seja repetido, uma vez confirmado usamos novamente uma condição *if* que verifica se o valor de *distanceFilms* é inferior ao de *Jacmax1*. Usamos o *distanceFilms* para que possamos calcular a distância de Jaccard entre os filmes com os 2 *ids.* Após verificada a condição atribuímos à variável id1 o id do *simFilm*, atribuímos também um novo valor a *Jacmax1* através do *dinstancesFilms* que é uma matriz criada no script 1 através da função *getDistancesMinHashFilms()* explicada em baixo. Usando o comando *distancesFilms(filme\_atual,simFilm)* vamos buscar à matriz a distância entre o filme atual e o filme similar.

Repetimos todos os ciclos e todas as condições e acrescentamos ainda mais uma verificação através de uma condição *if* que verifica se o *ID* do novo filme Similar (*SimFilm*) não é igual ao do marcado anteriormente.

A variável matrizMinHashFilms é uma matriz minHash que é inicializada através da função minHashFilms que recebe como argumentos *films, numHash e userE* que são os ID’s dos filmes, o número de hash functions que vamos usar e os ID’s dos utilizadores que viram cada filme, respetivamente, e retorna a matriz minHash dos filmes.

No script1 inicializamos a função *getDistancesMinHashFilms* com os valores recebidos, o *numFilms,* matrizMinHashFilms e numHash*.*



Figura 6 - variáveis opção 2 file2

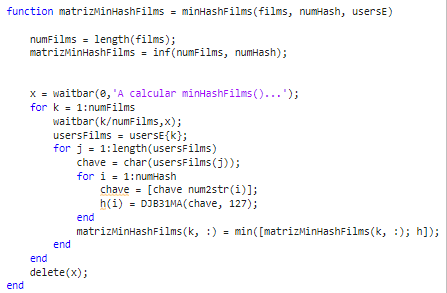


Figura 7 - função minHasFilms()

A função recebe de argumentos de entrada a variável *films,* que contém a lista de ids dos filmes, *numHash* é o número de hash functions que vamos executar e por fim a variável *userE,* que nos indica a lista de ids dos utilizadores que avaliaram o filme. Criamos uma waitbar apenas para acompanhamento da progressão da execução da função em questão. Criamos um ciclo *for* com valor *k* que percorre de 1 ao número total de filmes, *userFilms* é a lista de *IDs* de utilizadores que viram o filme *k.* Um novo ciclo *for* com valor j é percorrido até ao valor do número de ids de utilizadores. Ao valor de *chave* é atribuído o valor do *ID* de cada utilizador para cada filme visto. Com um ciclo *for* iteramos sobre todas as *Hash* functions que vamos usar (100). Na variável chave guardamos primeiramente a chave e depois o número da iteração da *Hash* function. Para cada função de dispersão usamos a função *DJB31MA,* função cedida na aula. Na matriz minHashFilms atribuímos o valor mínimo desta mesma matriz. No final deste *for loop* esse valor é armazenado na matriz *matrizMinHashUsers* , apenas se for menor do que o lá guardado anteriormente. Assim que terminada apagamos a *waitbar* com o código *delete*.

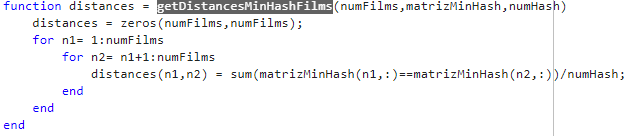


Figura 8 - função getDistancesMinHashFilms()

A função *getDistancesMinHashFilms ()* inicializa a variável *distances* que é uma matriz de zeros de tamanho *numfilms por numFilms.* A seguir fazemos um ciclo *for* para percorrer toda a matriz linha por linha, coluna por coluna onde atribuímos à linha n1, coluna n2, a distância de *Jaccard* entre n1, n2.

*O evalUsers1 é a lista dos users que viram o filme 1 e o evalUsers2 é a lista dos users que viram o filme 2. Para saber o número de users que viram o filme um e o filme dois usamos a função height* que retorna o número de linhas de *numevalUsers1temp e numevalUsers2temp respetivamente.*



Figura 9 - variáveis da opção 2 file2

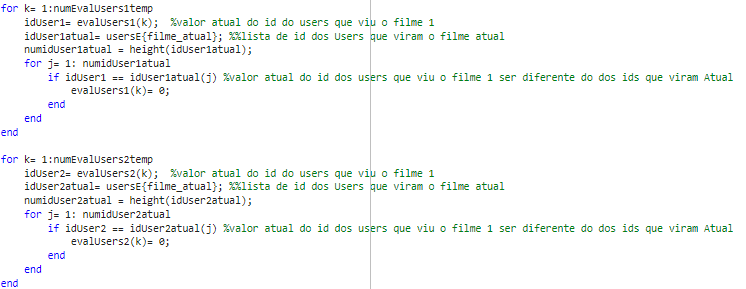


Figura 10 - procura de 2 filmes similares

Num ciclo *for* a variável *k* é iterada desde 1 até ao valor de *numEvalUsers2temp*, atribuímos à variável *idUser1* o valor de *evalUsers1* por cada filme, ou seja, o valor atual(k) d*o id* dos users que viram o filme 1. A variável *idUser1atual* corresponde à lista de *IDs* dos users que viram o filme atual. Para sabermos o número de users que viram o filme atual usamos a função do matlab height. Percorremos um novo ciclo *for* desde 1 até a numUser1atual através da variável j e com a condição *if* verificamos se o id do user *j* que viu o filme atual é igual ao id dos que viram o filme 1. Após se verificar atribuímos o valor de 0 ao id do User que viu o filme atual. Desta maneira podemos sinalizar os users que viram o filme 1 e o filme atual. Repetimos exatamente o mesmo processo, mas desta vez para os users que viram o filme 2.

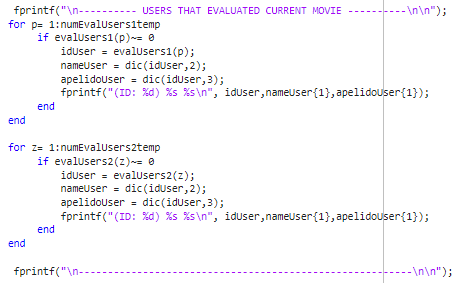


Figura 11 - print da opção 2

Fazemos um ciclo for usamos a variável *numEvalUsers1temp* inicializada anteriormente para percorrer todos os users que viram o filme 1. Usamos uma condição *if* para verificarse o valor da variável *evalUsers1* não está a 0 (indica que o user viu o filme atual e o filme similar 1), caso se confirme guardamos o ID do user atual, o seu nome próprio e o apelido nas variáveis *evalUsers1, nameUser e apelidoUser ,*respetivamente e executamos um print destes valores.

## Opção 3

Na opção 3 tivemos alguns problemas na execução das funções do script 1, sendo o erro na linha 136 ao executarmos o código “*unique”,* obtemos o erro apresentado na figura 13. Este erro deve-se ao facto de existirem células do *cell array* *interesses* que não deveriam ser adicionadas pois são células vazias no *dic,* mesmo executando o código da linha 130 não foi possível obtermos um array apenas de Strings para que se possa usar a função *unique,* que nos levaria a ter apenas um array de strings únicas, ou seja uma lista de cada interesse apresentado uma única vez.

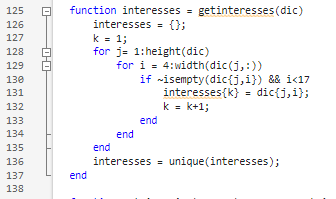


Figura 12 - getInteresses()

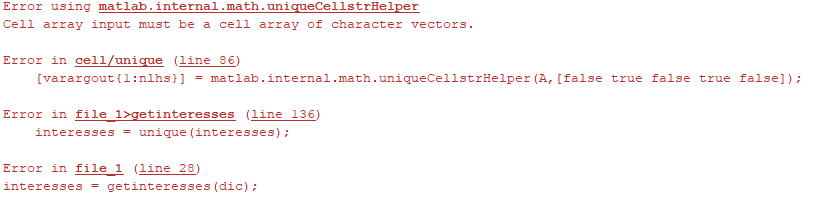


Figura 13 - erro do unique()

Posto isto decidimos comentar as chamadas às funções relativas às opções para que as restantes opções possam ser executadas. Porém iremos na mesma prosseguir à explicação do código desenvolvido para esta opção.

Escolhemos o valor de 100 funções de dispersão e atribuímos esse valor à variável *numHash*. Para obtermos a lista de todos os interesses, sem repetições, chamamos a função *getinterresses* já apresentada anteriormente na figura 12. Esta função é inicializada com o parâmetro de entrada *dic,* que é um array com diversas informações sobre os users, inicializamos uma variável *interesses* com o valor de um array vazio e uma *k* com valor 1. Executamos um ciclo *for com* a variável *j* que vai percorrer todas as linhas do dicionário, de seguida fazemos outro *for* mas desta vez percorremos desde 4 (célula a partir do qual aparecem os interesses do utilizador) até à largura da linha em questão, ou seja todas as colunas dessa linha. Executamos uma condição *if* na tentativa de verificar quando uma célula do *dic* que estamos a percorrer está vazia e desta maneira o seu valor não ser adicionada aos valores do array *interesses*, e também verificamos se o valor das colunas não excede 17 que é a barreira máxima do *dic.* Se a celula não estiver vazia e a coluna não exceda o valor 17 adicionamos ao array interresses no indice *k* o valor da presente célula de *dic.* No fim incrementamos o valor de *k* para que seja adicionado um novo intereresse numa nova posição do array. Por fim já fora de todos os ciclos *for* tentamos usar a função *unique* na tentativa de obtermos um array sem repetições de interesses e por fim obtermos a lista final que necessitamos. A variável *numInteressses* inicializada pelo valor total do número de interesses unicos existentes.

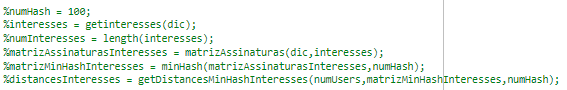


Figura 14 - variáveis opção 3 file1

Para criar a matriz de assinaturas dos interesses dos ultilizadores (*matrizAssinaturasInteresses*) criámos a função *matrizAssinaturas* que recebe como parâmetros de entrada o *dic* e a variável *interesses.*  A *matrizMinHashInteresses* é uma matriz minHash que é inicializada através da função minHashFilms que recebe como argumentos a *matrizAssinaturasInteresses*  e o *numHash.* Por fim temos a variável *distancesInteresses* que guarda as distâncias entre os interesses através da função *getDistancesMinHashInteresses,* em que os parâmetros de entrada são o número de utilizadores*(numUsers),* a matriz minHash de interesses (*matrizMinHashInteresses*) e o número de hash functions(*numHash*).

Todas as funções vão ser explicadas a seguir.

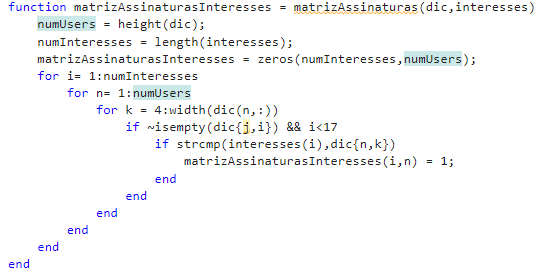


Figura 15 - função matrizAssinaturas()

Já dentro da função, começamos por criar 3 variáveis, a *numUsers* que guarda o número total de utilizadores através da função *height* do *matlab* que retorna o número de linhas desse *array,* a *numInteresses* que através da função length guarda o número de interesses e por fim a *matrizAssinaturasInteresses* que é uma matriz inicializada toda com zeros com *numInteresses* linhas e *numUsers* colunas. De seguida fizemos um ciclo *for* para percorrer todos os interesses que estão guardados na variável *interesses*, depois outro ciclo *for* para percorrer todos os utilizadores, mais um ciclo for para percorrer as colunas onde estão os interesses no *dic*, ou seja, noda coluna 4 até ao número máx de colunas (width(dic(n,:))). Seguidamente, verificámos se a célula do *dic* onde estamos não está vazia e também se o *i* não passa dos 17 que é o número máx de colunas.

Dentro deste *if* usámos mais uma condição *if* para verificar se encontramos um interesse da lista *interesses* numa célula do dic, caso se verifique, acrescentamos á matriz *matrizAssinaturasInteresses* o valor de 1 na posição (*i,n*) ou seja linha *i* coluna *n* para sinalizar a ativação do Interesse da linha i e coluna n.

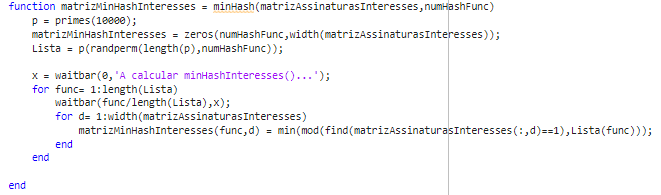


Figura 16 - função minHash()

A função *minHash* tem como objetivo calcular o valor de *minhash* para uma matriz de dados de gêneros de filmes e um número de funções de dispersão a usar. É inicializada com 2 parâmetros de entrada a *matrizAssinaturasInteresses* e *numHashFunc*, o número de funções de dispersão que vamos usar. Selecionamos números primos aletorios e, em seguida, calculamos o módulo do índice de cada elemento igual a 1 na matriz dados pelo número primo atual. O valor mínimo desses módulos é armazenado na matriz *matrizMinHashInteresses* e a função retorna essa matriz como resultado. Esta é usada para calcular a similaridade entre dois conjuntos de elementos de maneira eficiente. Durante a execução da função criamos uma waitbar apenas para acompanhamento da progressao da execussao da função em questão.

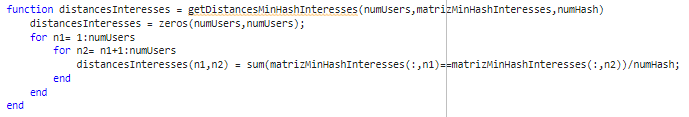


Figura 17 - função getDistancesMinHashInteresses()

Na função *getDistancesMinHashInteresses* inicializamos uma variável *distancesInteresses* que é uma matriz de zeros com um numéro de linhas e colunas igual ao número de utilizadores. A seguir fizemos um ciclo for, para percorrer todos os utilizadores, que começa em 1 e outro *for loop* que está sempre uma unidade à frente do anterior. Por fim a distância de *Jaccard* entre cada interesse é calculada.

Na opção 3 inicializamos a variável *evalUsers* guardamos o ID do utilizador que avaliou o filme atual, e depois chamamos a função *searchInteresses* onde passamos como argumentos de entrada as variáveis *evalUsers , matrizMinHashInteresses* , *numHash e* *interesses.*



Figura 18 – opção 3

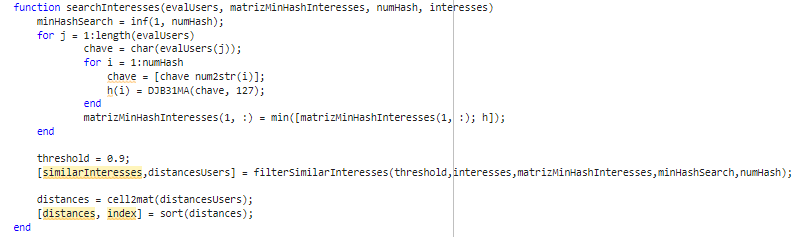


Figura 19 - função matrizMinHashInteresses()

A função *searchInteresses* inicializada com os parâmetros de entrada evalUsers, matrizminHashInteresses, explicados anteriormente, numHash, número de hash functions e a lista de interesses, *interesses*. Primeiramente calcula os valores *minHash* como referido anteriormente, e em seguida, usa a função *filterSimilar* para encontrar interesses na matriz de valores *minHash* que tenham uma alta similaridade com o interesse em questão. Atribuimos o valor de 0.9 à variavél *threshold* que é o limite máximo da distância de Jaccard pedida. Através da função *filterSimilar encontramos os interesses com maior similaridade com o referido.*

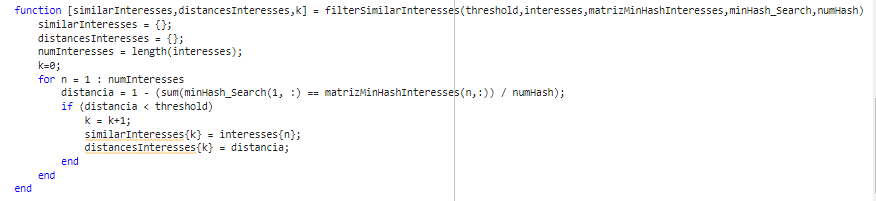


Figura 20 - filtersimilarinteresses()

A função *filterSimilarInteresses* tem como parametros de entrada *threshold, titles, matrizMinHashTitles, minHash\_Search e numHash,* explicados anteriormente. Inicializamos os arrays similarInteresses e distanceInteresses, atribuimos à variável numInteresses o número de interesses e à variável *k* o valor 0. Recorremos a um ciclo for para percorrer todos os interesses e calculamos a distância através da fórmula da distância de *Jaccard.* Na condição *if* verificamos se a distancia é menor que o *threshold* pedido (0.9), caso se confirme, *k* que serve como índice, é incrementado e no *similarInteresses* com índice *k* guardamos o interesse atual, e na *distanceInteresses* guardamos a distância.

Por fim era suposto ser apresentado os *IDs* e os nomes dos dois utilizadores que aparecem no maior número de conjuntos. Porém visto que não foi possivel executar as funções inicialmente não podemos completar esta opção.

## Opção 4

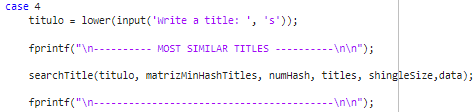


Figura 21 - opção 4

Na opção 4 é pedido ao utilizador para que insira uma string com o nome de um título (ou parte de um nome). Posto isto atribuímos à variavel *input a string, em minusculas, inserida pelo utilizador. Para executarmos a procura desses filmes usamos a função searchTitle(figura 25).*

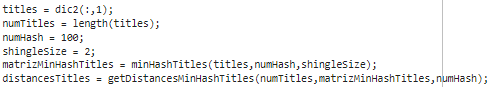


Figura 22 - variáveis opção 4 file1

Inicializamos no script 1 a variável *titles* que guarda todos os títulos do *dic2,* a variavel *numTitles* que guarda o número total de titles, a numHash que é o número de funções hash, o shingleSize que guarda o tamanho do shingle usado para calcular os valores minHash para o termo de pesquisa, a matrizMinHashTitles que é a matriz minHash dos *titles* e a *distanceTitles* que guarda a distância de Jaccard dos titles através da função *getDistancesMinHashTitles.*

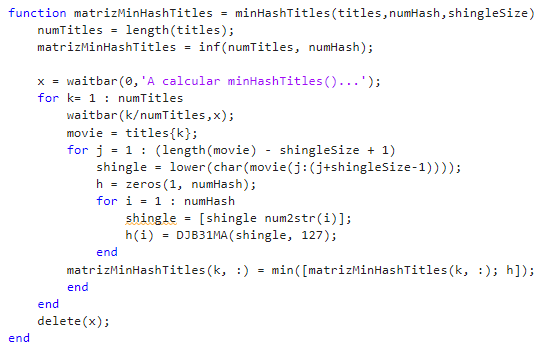


Figura 23 - minHashTitles()

A função *minHashTitles* à semelhança na função *minHash* usada anteriormente começa por calcular os MinHash, desta vez, dos títulos dos filmes, usando a matriz *matrizMinHashTitles* para os registar. Para isso, definimos o número de funções de dispersão igual a 100 e o tamanho dos shingles igual a 2. O raciocínio para serem calculados foi semelhante ao da alínea anterior.

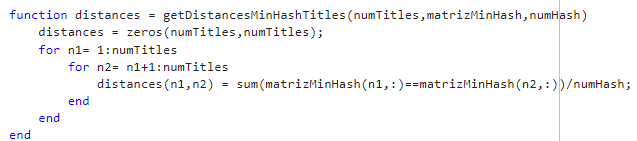


Figura 24 - função getdistancesMinHashtitles()

A função *getDistancesMinHashTitles* é semelhante à *getDistancesMinHashFilms,* só mudam os parâmetros de entrada, que neste caso são o número de títulos, a matriz *minHash* dos títulos e o número de *hash* functions.



Figura 25 - função searchTitles()

Nesta função usada parcialmente anteriormente e com algumas mudanças tem como função ser uma função de pesquisa. As mudanças começam no primeiro ciclo for em que temos de ter em conta o tamanho do Shingle (*shingleSize*). O threshold é inicializado com 1 uma vez que não nos dizem a distância máxima de Jaccard. Usamos de seguida a função *filterSimilar* para a busca dos títulos mais similares em questão.

As variáveis *numData* e *numTitles* correspondem ao número de linhas do *array data* e ao número de titulos total respetivamente. Temos um ciclo *for* que vai de 1 a 3 para percorrer os 3 títulos mais similares, dentro deste inicializamos a variável *count* que depois vai ser utilizada para contar quantos utilizadores avaliaram o filme com nota superior ou igual a 3. Recorremos a mais um ciclo *for* para percorrer os *titles*, dentro deste temos uma condição *if* que serve para saber o ID do filme similar. Depois usamos mais um ciclo for para percorrer todas as linhas do *data* e com a condição *if* procuramos o *ID j* na segunda coluna do *data* que tem todos os *ID’s* dos filmes, quando o encontrarmos verificamos através de outro *if* se a avaliação desse filme for maior ou igual a 3(a avaliação do filme está guardada na coluna 3 do *data*) incrementamos o *count*. No final damos *print* dos filmes mais similares e do número de vezes que esse filme foi avaliado com uma nota igual ou superior a 3.

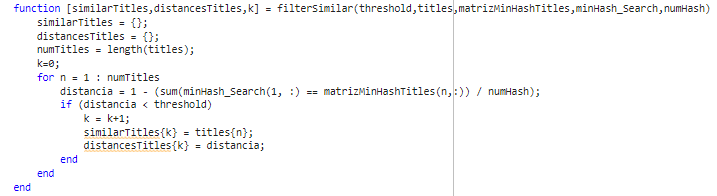


Figura 26 - função filterSimilar

A função filterSimilar serve para encontrar títulos na matriz de valores minHash que tenham uma alta similaridade com o termo de pesquisa. Ela é muito parecida com a *filterSimilarInteresses* utilizada anteriormente, o que muda são os parâmetros de entrada.

# Escolha de Valores

Testamos diversas opções desde 250 funções de dispersão, porém, o código demorava grandes períodos de tempo a correr. De seguida, reduzimos para 150 funções, mas o problema era o mesmo. Decidimos, assim, utilizar 100 funções de dispersão para todas as opções, por uma questão de otimização, uma vez que para este valor, o tempo de espera não era tão elevado. Ainda assim, 100 função de dispersão fornece uma boa precisão para a distância de Jaccard.

Quanto ao tamanho dos *shingles*, entre os valores pedidos (2 e 5) 3 é um tamanho adequado, pois quanto maior for o tamanho do *shingle,* menor vai ser a probabilidade de haver correspondência com outros *shingles*, isto é, é mais improvável obtermos dois *shingles* iguais, posto isto excluimos o tamanho igual a 2. Após algumas tentativas para valores de 3 e igual a 4, concluímos que havia mais correspondências com 3, então usamos este valor para uma mais correta funcionalidade do programa.

# Conclusão

Com a realização deste projeto no âmbito da disciplina de MPEI Métodos Probabilísticos para Engenharia Informática) conseguimos reforçar os nossos conhecimentos quanto aos assuntos abordados no guião 4. Melhoramos a nossa utilização do Matlab e compreendemos ter alcançado grande parte dos objetivos propostos ainda que em alguns pontos não tivesse sido possível progredir devido a um erro presente do código, o qual não foi possível resolver.