1 Introdução ao MATLAB

O MATLAB é um pacote de *software* comercial cujo nome deriva de `MATrix LABoratory`. Começou por ser um programa iterativo para cálculos com matrizes, tal como o nome indica, e transformou-se numa linguagem matemática de alto nível. O seu desenvolvimento permite agora, por exemplo, a resolução de equações diferenciais, o desenho de gráficos a duas e três dimensões e a resolução problemas de otimização. Possuindo o MATLAB uma linguagem de programação, qualquer algoritmo pode ser implementado em MATLAB.

Pode ser obtida ajuda no MATLAB sobre qualquer função digitando apenas na janela de comandos a palavra 'help' seguida do nome da função sobre a qual pretendemos obter informações.

Os comentários em MATLAB podem ser introduzidos precedidos do sinal `%`.

1.1 Variáveis

O MATLAB não requer nenhum tipo de declaração de variáveis ou dimensões. Quando um novo nome de variável é encontrado, o MATLAB automaticamente cria a variável e aloca a memória necessária. Se a variável já existir, o seu conteúdo é substituído pelo novo e, caso seja necessário, faz uma nova alocação de memória. O nome das variáveis é 'case sensitive', ou seja `a`$\neq$`A`, por exemplo. Os nomes das variáveis devem começar por uma letra, seguida de outras letras, dígitos ou `underscore`. Outros caracteres nunca devem ser usados. Não devem ser usados também nomes de variáveis que já estejam a ser usados pelo MATLAB, como por exemplo `exp`, `log`, ... As variáveis, por defeito, são matrizes, sendo que podem ser vetores ou escalares, já que estes são casos particulares de matrizes.

1.2 Números

O MATLAB usa a notação decimal convencional nos números, com um ponto decimal opcional, podendo o número ser precedido de mais ou menos. A notação científica usa a letra 'e' para especificar potência de 10. Os números imaginários usam 'i' ou 'j' como sufixo. Todos os números são armazenados internamente no formato especificado pela norma `IEEE floating-point`, que tem uma precisão finita de cerca de 16 algarismos significativos e uma gama finita de 10^{-308} a 10^{308}, aproximadamente.

O comando `format` define a forma como os números são apresentados, não tendo qualquer influência na forma como são armazenados. O formato que está definido por defeito é o `format short`, que apresenta os números com cinco dígitos e um ponto decimal fixo. O `format long` apresenta um formato de 15 dígitos e um ponto decimal fixo. Existem outros formatos disponíveis em MATLAB.

1.3 Funções, constantes e variáveis especiais

Apresenta-se de seguida uma lista de algumas funções, constantes e variáveis das mais usadas. Esta lista não pretende ser exaustiva, mas apenas um suporte para um utilizador inexperiente.

`ans` - variável *built in* criada quando não é atribuído um nome à variável

`pi` - valor de π (3.14159265...)

`i,j` - unidade imaginária (como sufixo)

`inf` - infinito

`NaN` - 'Not-a-Number'

`exp(x)` - exponencial de x (base neperiana)

`log(x)` - logaritmo natural de x

`log10(x)` - logaritmo na base 10 de x

`sqrt(x)` - raiz quadrada de x

`abs(x)` - valor absoluto de x

`rem(x,y)` - resto da divisão de x por y

`round(x)` - converte x para o valor inteiro mais próximo

`sin(x)` - seno de x

`cos(x)` - cosseno de x

`tan(x)` - tangente de x

`cot(x)` - co-tangente de x

1.4 Operações básicas com matrizes

Apresentam-se de seguida os operadores matemáticos mais usados, bem como algumas operações básicas com matrizes. As operações são sempre efetuadas segundo as regras de operações algébricas entre matrizes. Se se pretender efetuar uma operação para cada ponto da matriz, deve anteceder-se a operação em questão de '.'.

`+` adição

`-` subtração

`*` multiplicação

`/` divisão

`^` potenciação

`'` transposta de uma matriz

1.5 Vetores e matrizes

Uma das grandes vantagens do MATLAB é que considera a matriz como elemento básico, sendo que a sua dimensão não tem de ser definida *a priori*. Por esse motivo, permite a manipulação e criação de matrizes de uma forma muito mais rápida e intuitiva que em outras linguagens.

Um vetor ou uma matriz deve ser introduzido entre parêntesis retos, `[]`. Se a variável for um escalar, não é necessário colocar parêntesis.

Para se introduzir um vetor linha, os elementos devem ser separados por um espaço ou por uma vírgula.

```
>> u=[1 2 3]
```

ou

```
>> u=[1,2,3]
```

Um vetor linha cujos os elementos tenham uma distância constante entre si, podem ser criados de uma forma mais fácil usando ':'. Indica-se o primeiro elemento do vetor, de seguida o espaçamento entre os elementos e o elemento final do vetor (ou valor máximo até onde

se pretende que os elementos do vetor cheguem). Se o espaçamento for a unidade, pode suprimir-se o espaçamento.

Para se introduzir um vetor coluna, os elementos devem ser separados por ponto e vírgula, ou cria-se através de um vetor linha fazendo a sua transposta (usando ').

Para introduzir uma matriz combinam-se as regras anteriores.

Para se obter a dimensão de uma matriz, usa-se o comando `size`. Esta dimensão é devolvida como um vetor de dois elementos, em que o primeiro representa o número de linhas e o segundo o número de colunas. Para se obter o número de elementos num vetor usa-se o comando `length`. Este comando aplicado a uma matriz dá o valor máximo entre o número de linhas e o número de colunas.

1.6 Aceder a partes de uma matriz

Pode aceder-se a partes específicas de uma determinada matriz, bastando para isso indicar-se as coordenadas dos elementos aos quais se pretende aceder. As posições indicadas à esquerda da vírgula indicam as coordenadas das linhas, à direita da vírgula as coordenadas das colunas. Se quisermos incluir todas as linhas (ou todas as colunas) de uma matriz, de uma forma simplificada basta indicar ':' na respetiva posição.

1.7 Matrizes especiais

O MATLAB gera algumas matrizes de forma automática. A dimensão da matriz é indicada entre parêntesis, sendo que (n, m) cria uma matriz com n linhas e m colunas e (n) cria uma matriz quadrada $n \times n$.

`>>rand(m,n)` ou `>>rand(n)` - gera uma matriz com números aleatórios entre zero e um.

`>>eye(n,m)` ou `>>eye(n)` - gera uma matriz com os elementos da diagonal principal iguais a um e os restantes iguais a zero. No caso da matriz ser quadrada, o comando `eye` gera a matriz identidade.

`>>zeros(m,n)` ou `>>zeros(n)` - gera uma matriz com todos os elementos iguais a zero.

`>>ones(m,n)` ou `>>ones(n)` - gera uma matriz com todos os elementos iguais a um.

`>>magic(n)` - gera uma matriz em que a soma dos elementos de cada linha e a soma dos elementos de cada coluna é sempre igual.

1.8 Ficheiros m

Os ficheiros usados em MATLAB devem ter a extensão `m`. Os mais usados são os que definem funções, com parâmetros de entrada e de saída, e os do tipo `script`, que executam um conjunto de comandos sequencialmente, pela ordem que aparecem escritos.

Para se criar um novo ficheiro, a forma mais fácil é usar o editor do MATLAB, que no caso das funções já aparece por defeito com a estrutura definida. Um ficheiro do tipo função começa sempre pela palavra `function` na primeira linha. De seguida define-se a estrutura da função. Primeiro aparecem os parâmetros de saída (se for mais que um terão de aparecer entre parêntesis retos, []), seguidos do sinal '=', o nome da função, que deve coincidir com o nome do ficheiro e os parâmetros de entrada, que aparecem entre parêntesis curvos, (). Nas linhas seguintes aparecem as relações funcionais entre os parâmetros de entrada e os parâmetros de saída.

2 Método EGPP

O MATLAB permite resolver sistemas lineares, calcular a inversa e o determinante de uma matriz quadrada, através de EGPP. Sendo `A` a matriz dos coeficientes e `b` o vetor dos termos independentes, os comandos a usar são `A\b`, para resolver o sistema linear correspondente, `inv(A)` para calcular a inversa de `A` e `det(A)` para calcular o determinante de `A`.

3 Equações não lineares

Para resolver equações não lineares o MATLAB tem três funções: a função `roots`, que calcula os zeros de um polinómio, a função `fzero`, que calcula a solução de qualquer equação não linear através de um método do tipo bisseção, e a função `fsolve`, que permite não só calcular a solução de equações não lineares, como também calcular a solução de sistemas de equações não lineares. Por esta última função ser mais abrangente, vai descrever-se de seguida.

3.1 fsolve

A função `fsolve` pode ser usada para resolver equações não lineares ou sistemas de equações não lineares, fornecendo-se ou não as primeiras derivadas da função ou funções. A sua sintaxe é

```
[x,f,exitflag,output] = fsolve('func',x0,options)
```

em que os parâmetros de saída definem a solução em `x`, o valor da função na solução `f` (que deverá ser próximo de zero), a `exitflag` define a forma como parou o algoritmo (a desejável é 1 - significa que o processo convergiu para uma raiz) e na estrutura `output` encontra-se informação sobre o processo iterativo (número de iterações, número de cálculos da função, algoritmo usado...). Os parâmetros de entrada a fornecer são a equação ou sistema de equações a resolver, que devem ser escritos numa `m-file` do tipo função. Opcionalmente, esta `m-file` poderá também conter as primeiras derivadas da função ou funções. `x0` é o valor inicial e na estrutura `options` podem alterar-se alguns dos valores que o MATLAB tem por defeito, nomeadamente definir se é pretendido usar as derivadas fornecidas na `m-file`. Para se saber que opções podem ser alteradas e quais os valores que se encontram por defeito no MATLAB, basta escrever na janela de comandos `fsolve('defaults')`. Para se alterar a estrutura `options` deve usar-se a sintaxe

```
options = optimset('param1',value1,'param2',value2,...)
```

Os valores que são *strings* devem indicar-se entre plicas (').

Exemplo:

Resolver a equação não linear $\sin(3x) + 4x^2 = 0$, fornecendo a primeira derivada. Primeiro cria-se uma `m-file` que contenha a função e a derivada.

Resolução:

```
function [f,d]=exEQNL(x)
f=sin(3*x)+4*x^2;
if nargout>1
    d=3*cos(3*x)+8*x;
end
```

O comando `if` aparece para tornar a função mais eficiente. Sempre que o `fsolve` precisar

de recorrer ao cálculo da função, só passa ao cálculo da derivada caso isso seja solicitado. Se não se colocar o `if`, sempre que se aceder à função será calculada a função e a derivada.

Para que seja usada a derivada, não basta colocá-la na `m-file`, é necessário indicar nas opções que se pretende usá-la, e isso faz-se com a opção `Jacobian`. Assim, a primeira coisa a fazer é alterar a estrutura das opções:

```
>>op=optimset('Jacobian','on');
```

A sequência de comandos deverá ser

```
>>x0=5;
>>[x,f,extf,outp]=fsolve('exEQNL',x0,op)
```

4 Interpolação polinomial

4.1 polyfit

A função `polyfit` ajusta um polinómio de grau n a um conjunto de dados, no sentido dos mínimos quadrados. No entanto, pode ser usado para determinar um polinómio interpolador, sendo que, neste caso, terão de ser fornecidos exatamente $n+1$ pontos, que devem ser escolhidos de forma adequada, como foi descrito nas secções anteriores deste capítulo.

A sintaxe desta função é

```
p = polyfit(x,y,n)
```

em que `p` devolve os coeficientes do polinómio em potências descendentes. `x` e `y` são dois vetores que contêm os dados que se pretendem aproximar e `n` é o grau do polinómio.

4.2 polyval

Para se fazer a estimativa obtida pelo polinómio interpolador calculado num ponto, ou num conjunto de pontos, usa-se a função `polyval`, cuja sintaxe é

```
Y=polyval(p,X)
```

em que `p` é o vetor que contém os coeficientes do polinómio obtidos através da função `polyfit` e X é o valor ou um vetor contendo um conjunto de valores onde se pretende fazer a estimativa usando o polinómio calculado.

5 splines

Para se obter uma spline cúbica em MATLAB, usa-se o comando `spline`. A sintaxe é

```
PP = spline(x,y)
```

em que `PP` vai conter uma estrutura que em `coefs` terá uma matriz de quatro colunas com os coeficientes que permitem escrever cada um dos segmentos da spline. Cada linha da matriz corresponde a um segmento, sendo que cada coluna representa os coeficientes do polinómio de grau três por ordem decrescente, na forma

$$c_1(x-x_0)^3 + c_2(x-x_0)^2 + c_3(x-x_0) + c_4,$$

sendo x_0 o nó inicial do segmento em questão.

Não existe nenhum comando em MATLAB para o cálculo de splines cúbicas naturais. No entanto, podem calcular-se splines cúbicas completas, indicando o declive (primeira derivada) nos extremos. A sintaxe, neste caso, passa a ser

```
PP = spline(x,[dd0 y ddn])
```

em que `dd0` e `ddn` são as derivadas no ponto inicial e no ponto final, respetivamente.

Para se determinar a aproximação a um valor ou conjunto de valores, usa-se o comando `spline`, mas indicando como terceiro parâmetro de entrada o valor do ponto (ou vetor de pontos) interpolador:

```
YY = spline(x,y,XX)
```

ou

```
YY = spline(x,[dd0 y ddn],XX)
```

Em alternativa pode usar-se o comando `ppval`:

```
YY = ppval(PP,XX)
```

6 Integração numérica

6.1 trapz

A função `trapz` do MATLAB implementa a fórmula do trapézio para resolver um integral numericamente, com base num conjunto de pontos (x, y). A sintaxe é

```
>>I=trapz(x,y)
```

`I` é a aproximação ao integral e `x` e `y` são vetores que contêm os dados e têm de ter a mesma dimensão.

6.2 integral

A função `integral` do MATLAB implementa quadratura global adaptável. É necessário fornecer a função a integrar (com o comando `inline` ou criando uma `m-file` do tipo função). Para se usar a função `quad` é necessário conhecer-se a expressão analítica da função integranda. Caso seja apenas conhecida uma tabela de pontos, só pode ser usada a função `trapz`. A sintaxe é

`I=integral('func',a,b,'AbsTol',val1,'RelTol',val2)`.

É devolvida a aproximação ao integral em `I`. `func` é o nome da função a integrar, `a` e `b` são os limites de integração e opcionalmente podem fornecer-se valores para a tolerância absoluta `AbsTol` e/ou para a tolerância relativa `RelTol`. Os valores por defeito são 10^{-10} e 10^{-6}, respetivamente. Quanto maiores os valores de `AbsTol` e `RelTol`, menor é o tempo de computação. No entanto os resultados serão menos exatos.

7 Mínimos quadrados

7.1 polyfit e polyval

À semelhança do que se fez para aproximar polinómios interpoladores, pode usar-se a função `polyfit` para estimar modelos polinomiais no sentido dos mínimos quadrados. Neste caso usam-se todos os pontos disponíveis. A função pode devolver, como segundo parâmetro de saída, a norma do resíduo, pelo que para se obter o resíduo este valor terá de ser elevado ao quadrado.

```
[p,r] = polyfit(x,y,n)
```

`p` devolve os coeficientes do polinómio em potências descendentes e `r` uma estrutura que contém no seu último termo a norma do resíduo (`normr`). `x` e `y` são dois vetores que contêm os dados que se pretendem aproximar e `n` é o grau do polinómio.

Para se fazer a estimativa obtida pelo polinómio interpolador calculado num ponto, ou num conjunto de pontos, usa-se a função `polyval`, tal como foi descrito na Subsecção 4.2.

7.2 lsqcurvefit

A função `lsqcurvefit` resolve qualquer problema de mínimos quadrados. Como tal, é adequada para problemas não polinomiais. A sua sintaxe é

`[c,S] = lsqcurvefit('mq',c0,x,y)`.

`c` é o vetor que contém os parâmetros do modelo e `S` é a soma dos quadrados dos resíduos. `mq` é uma `m-file` que contém o modelo que se pretende estimar, `c0` é o vetor com os valores iniciais dos parâmetros (caso não seja dito nada deve considerar-se um vetor de uns) e `x` e `y` são os vetores que contêm a tabela de dados a aproximar.

8 Métodos do gradiente

8.1 fminunc

A função `fminunc` encontra o mínimo de uma função multidimensional usando métodos do gradiente. A sua sintaxe é

`[xmin,fmin,exitflag,output] = fminunc('func',x0,options)`

em que os parâmetros de saída definem o minimizante em `xmin`, o valor mínimo da função `fmin`, a `exitflag` define a forma como parou o algoritmo (a desejável é 1 - significa que o processo convergiu para um ponto estacionário) e na estrutura `output` encontra-se informação sobre o processo iterativo (número de iterações, número de cálculos da função, algoritmo usado...). Os parâmetros de entrada a fornecer são a função que se pretende minimizar (caso o problema seja de maximização tem de se trocar o sinal da função *a priori*), que deve ser escrita numa `m-file` do tipo função. Opcionalmente, esta `m-file` poderá também conter as primeiras e/ou segundas derivadas da função. `x0` é o valor inicial e na estrutura `options` podem alterar-se alguns dos valores que o MATLAB tem por defeito, nomeadamente definir se é pretendido usar as derivadas fornecidas na `m-file`. Para se saber que opções podem ser alteradas e quais os valores que se encontram por defeito no MATLAB, basta escrever na janela de comandos `fminunc('defaults')`. Para se alterar a estrutura `options` deve usar-se a sintaxe

```
options = optimset('param1',value1,'param2',value2,...)
```

Os valores que são *strings* devem indicar-se entre plicas (').

9 Método de Nelder-Mead

9.1 fminsearch

A função `fminunc` encontra o mínimo de uma função multidimensional usando o método de Nelder-Mead. A sua sintaxe é

```
[xmin,fmin,exitflag,output] = fminsearch('func',x0,options)
```

em que os parâmetros de saída definem o minimizante em `xmin`, o valor mínimo da função `fmin`, a `exitflag` define a forma como parou o algoritmo (a desejável é 1 - significa que o processo convergiu para um ponto estacionário) e na estrutura `output` encontra-se informação sobre o processo iterativo (número de iterações, número de cálculos da função, algoritmo usado...). Os parâmetros de entrada a fornecer são a função que se pretende minimizar (caso o problema seja de maximização tem de se trocar o sinal da função *a priori*), que deve ser escrita numa `m-file` do tipo função. `x0` é o valor inicial e na estrutura `options` podem alterar-se alguns dos valores que o MATLAB tem por defeito. Para se saber que opções podem ser alteradas e quais os valores que se encontram por defeito no MATLAB, basta escrever na janela de comandos `fminsearch('defaults')`. Para se alterar a estrutura `options` deve usar-se a sintaxe

```
options = optimset('param1',value1,'param2',value2,...)
```

Os valores que são *strings* devem indicar-se entre plicas (').