Splines Cúbicos - JVS36

António Arco (nº89645), Iara Figueiras (nº89668) e Leonardo Pedroso (nº89691) – Grupo 19
Professor orientador: Prof. José Viriato Santos

***Resumo* — Com este projeto, pretendemos explorar as propriedades e aplicações dos splines cúbicos. Para tal, exploraremos a sua teoria basilar e algumas das diversas condições de fecho. Estudaremos, também, o erro induzido na interpolação, com especial ênfase na sua ordem de convergência. Investigaremos, ainda, um método adaptativo simples que comparamos com a interpolação por espaçamento uniforme, no anexo D. Por fim, aplicaremos os métodos desenvolvidos a alguns exemplos, em MATLAB.**

I. INTRODUÇÃO

O spline surge como uma alternativa à interpolação polinomial, como solução ao problema da interpolação de um conjunto de pontos. Com efeito, esta técnica consegue alcançar um erro reduzido e evitar fenómenos indesejáveis, utilizando polinómios de grau reduzido. Na verdade, uma das contrapartidas da interpolação polinomial é o facto de não convergir necessariamente para a função interpolada, conhecido como o fenómeno de Runge, em que o polinómio interpolador, a partir de certo grau, desenvolve oscilações, enquanto que na interpolação por um spline a convergência é garantida.

Em particular, um spline cúbico é composto por troços de polinómios de terceiro grau, com continuidade na função, na primeira e segunda derivadas na transição entre troços. Podemos, ainda, obter vários tipos de splines cúbicos, consoante as condições de fecho, alguns dos quais serão estudados nas subsecções seguintes.

Suponhamos que pretendemos interpolar os pontos

$$(x_0, y_0), (x_1, y_1), ..., (x_n, y_n) \tag{1}$$

Comecemos por condicionar a continuidade na segunda derivada, escrevendo explicitamente a segunda derivada do troço i do spline, S_i'', na forma de Lagrange,

$$S_i'' = M_{i-1}\frac{x_i - x}{h_i} + M_i\frac{x - x_{i-1}}{h_i} \tag{2}$$

em que M_i é o valor da segunda derivada do spline interpolador no nó x_i e $h_i = x_i - x_{i-1}$.
É de notar que vem de (2) que $S_i''(x_i) = S_{i-1}''(x_i)$, o que assegura a continuidade na segunda derivada para os nós interiores, i.e., para os nós $x_1, ..., x_{n-1}$.

Primitivando duas vezes a equação (2), por forma a obter a expressão explícita para o troço i, vem:

$$S_i = M_{i-1}\frac{(x_i - x)^3}{6h_i} + M_i\frac{(x - x_{i-1})^3}{6h_i} + c\frac{x_i - x}{h_i} + d\frac{x - x_{i-1}}{h_i} \tag{3}$$

onde c e d são constantes que advêm da integração e posterior representação na forma de Lagrange. Impondo as condições de continuidade na função, i.e.:

$$S_i(x_{i-1}) = y_{i-1} \qquad S_i(x_i) = y_i \tag{4}$$

vem que:

$$c = y_{i-1} - M_{i-1}\frac{h_i^2}{6} \qquad d = y_i - M_i\frac{h_i^2}{6} \tag{5}$$

Substituindo c e d em (3) temos a equação do spline para o troço i:

$$\begin{aligned} S_i = {} & M_{i-1}\frac{(x_i - x)^3}{6h_i} + M_i\frac{(x - x_{i-1})^3}{6h_i} \\ & + (y_{i-1} - M_{i-1}\frac{h_i^2}{6})\frac{x_i - x}{h_i} + (y_i - M_i\frac{h_i^2}{6})\frac{x - x_{i-1}}{h_i} \end{aligned} \tag{6}$$

Notamos que esta expressão depende da segunda derivada do spline interpolador em cada um dos nós, pelo que, a partir da condição de continuidade na primeira derivada, pretendemos obter uma relação entre essas constantes.

Derivando (6), obtemos a seguinte expressão:

$$\begin{aligned} S_i' = {} & -M_{i-1}\frac{(x_i - x)^2}{2h_i} + M_i\frac{(x - x_{i-1})^2}{2h_i} \\ & + \frac{y_i - y_{i-1}}{h_i} - (M_i - M_{i-1})\frac{h_i}{6} \end{aligned} \tag{7}$$

e tomando

$$S_i' = S_{i+1}'$$

vem que:

$$\begin{aligned} M_{i-1}\frac{h_i}{6} + M_i\frac{h_i + h_{i+1}}{3} + M_{i+1}\frac{h_{i+1}}{6} = \\ \frac{y_{i+1} - y_i}{h_{i+1}} - \frac{y_i - y_{i-1}}{h_i} \end{aligned} \tag{8}$$

para $i = 1, ..., n-1$

Em particular, para espaçamento uniforme h obtemos:

$$M_{i-1} + 4M_i + M_{i+1} = \frac{6}{h^2}(y_{i+1} - 2y_i + y_{i-1}) \tag{9}$$

Temos agora $n-1$ equações e $n+1$ incógnitas (os valores de M_i), pelo que são necessárias duas condições adicionais de fecho para a determinação do spline. Estas condições deverão ser selecionadas de acordo com o contexto em que o spline é usado. Exploraremos três tipos de spline cúbico nas subsecções seguintes.

A. Spline Natural

As condições de fecho do spline natural são:

$$M_0 = M_n = 0$$

Com estas novas identidades é fácil calcular os valores de M_i para $i = 0, 1, ..., n$ e, assim, determinar as equações de cada um dos troços.
Considerando o caso particular de espaçamento uniforme, usando (9), conseguimos reduzir o problema da interpolação por um spline cúbico à resolução de um sistema de equações lineares da forma:

$$A \begin{bmatrix} M_1 \\ \vdots \\ M_{n-1} \end{bmatrix} = B \tag{10}$$

com

$$A = \begin{bmatrix} 4 & 1 & & 0 \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ 0 & & 1 & 4 \end{bmatrix}.$$

$$B = \begin{bmatrix} \frac{6}{h^2}(y_2 - 2y_1 + y_0) \\ \vdots \\ \frac{6}{h^2}(y_n - 2y_{n-1} + y_{n-2}) \end{bmatrix}$$

em que h é o espaçamento entre nós adjacentes.
Assim, substituindo os valores de $M_0, M_1, ..., M_n$ na equação (6), obtemos o polinómio de terceiro grau de cada um dos troços. Da mesma forma, se pretendemos determinar a primeira e segunda derivadas do spline interpolador, substituímos estes valores em (7) e (2), respetivamente.

É de notar que a matriz A do sistema de equações lineares dado por (8) e condições fornteira do spline, para espaçamento não necessariamente uniforme, é tridiagonal, pelo que a solução pode ser obtida com uma menor complexidade computacional utilizando o algoritmo de Thomas em detrimento do método de eliminação de Gauss. Este método é estável uma vez que a matriz tridiagonal é diagonalmente dominante, i.e.:

$$A_{ii} \geq \sum_{j \neq i} |A_{ij}| \tag{11}$$

Por esta razão utilizaremos este método para sistemas lineares deste tipo, sempre que possível, implementado com o auxílio de [8].

B. Spline Completo

O Spline Completo tem como condições de fecho os valores da derivada da função interpolada nos nós extremos. Assim, é utilizado quando estas derivadas são conhecida. Temos, portanto:

$$S'(x_0) = y'_0 \qquad S'(x_n) = y'_n \tag{12}$$

Desta forma, usando estas condições em (7) obtemos as seguintes identidades:

$$2M_0 + M_1 = \frac{6}{h_i}(\frac{y_1 - y_0}{h_1} - y'_0) \tag{13}$$

$$M_{n-1} + 2M_n = \frac{6}{h_n}(y'_n - \frac{y_n - y_{n-1}}{h_n}) \tag{14}$$

Assim, considerando espaçamento uniforme, das equações (9), (13) e (14) podemos construir um sistema linear, que permite obter os valores de M_i para $i = 0, ..., n$, da forma:

$$A \begin{bmatrix} M_0 \\ \vdots \\ M_n \end{bmatrix} = B \tag{15}$$

com

$$A = \begin{bmatrix} 2 & 1 & & & 0 \\ 1 & 4 & \ddots & & \\ & \ddots & \ddots & \ddots & \\ & & \ddots & 4 & 1 \\ 0 & & & 1 & 2 \end{bmatrix}$$

$$B = \begin{bmatrix} \frac{6}{h}(\frac{y_1 - y_0}{h} - y'_0) \\ \frac{6}{h^2}(y_2 - 2y_1 + y_0) \\ \vdots \\ \frac{6}{h^2}(y_n - 2y_{n-1} + y_{n-2}) \\ \frac{6}{h}(y'_n - \frac{y_n - y_{n-1}}{h}) \end{bmatrix}$$

É de salientar que A, do sistema de equações lineares dado por (8) e condições fronteira do spline, para espaçamento não necessariamente uniforme, verifica (11), ou seja, é tridiagonal e diagonalmente dominante, pelo que o método de Thomas é, mais uma vez, estável.

C. Spline Not-A-Knot

As condições adicionais num spline Not-a-Knot são a imposição de continuidade na terceira derivada nos nós x_1 e x_{n-1}. Assim, nos troços 1 e 2, no ponto x_1, tanto a função como as primeira, segunda e terceira derivadas têm o mesmo valor, pelo que, sendo polinómios de terceiro grau, os troços S_1 e S_2 são dados pelo mesmo polinómio. Seguindo o mesmo raciocínio, concluímos que o mesmo sucede para os troços S_{n-1} e S_n. É esta particularidade deste tipo de spline que serviu de inspiração ao seu nome, uma vez que a função interpoladora é dada por um mesmo polinómio entre os nós x_0 e x_2 e entre x_{n-2} e x_n, deixando x_1 e x_{n-1} de desempenhar o papel de nós.

Por forma a condicionar a continuidade na terceira derivada nos nós x_1 e x_{n-1} obtemos as seguinte relações a partir da equação (2), fazendo $S''_1 = S''_2$ e $S''_{n-1} = S''_n$ para todo o x:

$$-\frac{M_0}{h_1} + \frac{M_1}{h_1} + \frac{M_1}{h_2} - \frac{M_2}{h_2} = 0 \tag{16}$$

$$-\frac{M_{n-2}}{h_{n-1}}+\frac{M_{n-1}}{h_{n-1}}+\frac{M_{n-1}}{h_n}-\frac{M_n}{h_n}=0 \tag{17}$$

Desta forma, munidos destas duas equações adicionais podemos calcular facilmente os valores de M_i para $i=0,1,...,n$ e, assim, determinar as equações de cada um dos troços, a partir de (6).

Se considerarmos o caso particular do espaçamento unifome, temos, das equações (16) e (17):

$$-M_0+2M_1-M_2=0 \tag{18}$$

$$-M_{n-2}+2M_{n-1}-M_n=0 \tag{19}$$

Conseguimos reduzir o problema da interpolação por um spline cúbico Not-A-Knot à resolução de um sistema de equações lineares da forma:

$$A\begin{bmatrix} M_0 \\ \vdots \\ M_n \end{bmatrix}=B \tag{20}$$

com

$$A=\begin{bmatrix} -1 & 2 & -1 & & & & 0 \\ 1 & 4 & 1 & & & & \\ 0 & 1 & 4 & \ddots & & & \\ & & \ddots & \ddots & \ddots & & \\ & & & \ddots & 4 & 1 & 0 \\ & & & & 1 & 4 & 1 \\ 0 & & & & -1 & 2 & -1 \end{bmatrix}$$

$$B=\begin{bmatrix} 0 \\ 6\frac{y_2-2y_1+y_0}{h^2} \\ \vdots \\ 6\frac{y_n-2y_{n-1}+y_{n-2}}{h^2} \\ 0 \end{bmatrix}$$

Para resolver este sistema de equações lineares, utilizaremos a operação $mldivide$ representada por \em MATLAB. Esta é uma função muito versátil e eficiente, pelo que a utilizaremos para a resolução de todos os sistemas de equações em que não seja possível aplicar o método de Thomas.

Uma vez que na grande generalidade as matrizes usadas neste projeto são de dimensões relativamente pequenas, não usaremos matrizes no formato *sparse*.

II. PROPRIEDADES

Nesta secção exploraremos algumas das propriedades da interpolação por splines cúbicos.

A. Teorema de Holladay (1957)

Deste teorema resulta que, considerando todas as funções $S(x)$ que interpolam um conjunto finito de pontos da forma em (1), o spline cúbico natural é a que minimiza

$$\int_{x_0}^{x_n}(S''(x))^2dx \tag{21}$$

Analisemos agora o porquê de esta ser uma propriedade desejável:

1) Sabendo que a curvatura de $S(x)$ é dada por $\frac{S''(x)}{(1+S'(x)^2)^{3/2}}$ é fácil verificar que o spline cúbico é a função interpoladora que a minimiza. Sendo por isso muito útil para, por exemplo, o planeamento de trajetórias de braços robóticos.
2) Temos ainda que a primeira aproximação para a energia potencial elástica de uma viga uniforme, delgada, elástica e linear é proporcional a $\int S''(x)dx$ em que $S(x)$ representa a o deslocamento vertical da viga em função da distância a uma das pontas, para um dado instante de tempo. Assim, pelo princípio da energia mínima $S(x)$ para uma viga apoiada é dada por um spline cúbico natural.

Podemos encontrar uma demonstração deste resultado em [5] e uma análise mais detalhada das suas aplicações em [6].

B. Propriedade da melhor aproximação

O sline completo é muitas vezes chamado spline interpolador ótimo. Na verdade, dada uma função $f(x)$, que pretendemos interpolar em nós da forma em (1), o spline completo $S(x)$ é aquele que minimiza:

$$\int_{x_0}^{x_n}(S''(x)-f''(x))^2dx \tag{22}$$

III. ANÁLISE DO ERRO

Nesta secção pretendemos estudar o erro cometido na interpolação por splines cúbicos. Na aplicação de aproximações a problemas de engenharia é sempre de extrema importância perceber qual a magnitude do erro introduzido, pelo que esta análise é fulcral.

A. Estimativa do majorante do erro

Temos de [1] e [2] que o majorante do erro cometido na interpolação por spline cúbico $S(x)$ de uma função $f(x)\in C^4$ é aproximado por:

$$||D^n(f(x)-S(x))||_\infty = c_n||D^4f(x)||_\infty h^{4-n} \tag{23}$$

em que:

$$c_0=\tfrac{5}{384}\quad c_1=\tfrac{\sqrt{3}}{216}+\tfrac{1}{24}\quad c_2=\tfrac{1}{12}+\tfrac{h}{3\underline{h}}\quad c_4=\tfrac{1}{2}(1+(\tfrac{h}{\underline{h}})^2)$$

onde definimos $h=\max(h_i)$ e $\underline{h}=\min(h_i)$.

Assim, para o caso particular de espaçamento uniforme h, temos que o majorante do erro na função é proporcional à quarta potência do espaçamento usado, i.e.:

$$||E||_\infty = ||f(x)-S(x)||_\infty \ \propto h^4 \tag{24}$$

de onde vem que:

$$\log||E||_\infty = k+4\log h \tag{25}$$

com $k\in\mathbb{R}$ constante. Esta equação permitirá avaliar a qualidade da estimativa da dependência, em h, do majorante do erro da função, para cada um dos tipos de spline. De forma análoga, partindo de (23) obtemos expressões semelhantes

para o erro na primeira e segunda derivadas, tomando $n = 1$ e $n = 2$, respetivamente.

B. Erro na função interpoladora para o spline completo

É possível, para um spline completo, deduzir um majorante do erro cometido na interpolação de uma função $f(x) \in C^4$ num dado ponto $x \in [x_i, x_{i+1}]$. Na verdade, temos de uma das expressões obtidas em [9] que:

$$|E(f,x)| \leq \frac{(x-x_i)^2(x_{i+1}-x)^2}{24} f^{(4)}(\xi_i) + \frac{(x-x_i)(x_{i+1}-x)}{24} \frac{||h||^3}{h_{i+1}} ||f^{(4)}||_\infty \tag{26}$$

em que $||h|| = \max(h_i)$ e $\xi_i \in [x_i, x_{i+1}]$. É fácil reparar que como

$$\frac{d}{dx}[(x-x_i)(x_{i+1}-x)] = -2x + x_{i+1} - x_i$$

se anula para

$$x^* = \frac{x_{i+1}+x_i}{2} \tag{27}$$

Dado que

$$\left.\frac{d^2}{dx^2}\right|_{x=x^*} [(x-x_i)(x_{i+1}-x) = -2 < 0$$

e a expressão se anula para os extremos do intervalo $[x_i, x_{i+1}]$ temos que (27) é máximo de $(x-x_i)(x_{i+1}-x)$ no troço i.
Desta forma podemos majorar (26) usando (27):

$$|E(f,x)| \leq \frac{h_{i+1}}{96}\left(\frac{h_{i+1}^3}{4} + ||h||^3\right)||f^{(4)}||_\infty \tag{28}$$

e, em particular, para espaçamento h uniforme:

$$|E(f,x)| \leq \frac{5}{384}||h||^4||f^{(4)}||_\infty \tag{29}$$

Vejamos que o resultado é consistente com (23) e ainda que, no caso particular do spline completo, essa desigualdade não se trata de uma aproximação, mas de um majorante exato.

IV. Resultados e sua interpretação na aplicação de alguns métodos

A. Interpolação de funções, e suas derivadas, por spline cúbico Not-a-Knot

Reparamos, pelas figuras 1, 2, 3, 4 e pelos gráficos no anexo A, que o spline not-a-knot, com espaçamento uniforme, constitui uma boa aproximação na interpolação de uma função e aproximação da sua primeira derivada, apresentado algumas dificuldades em extrapolar, através da segunda derivada do spline, a segunda derivada da função.
É especialmente importante notar que o erro é significativamente superior em troços em que a quarta derivada da função interpolada é elevada. Este efeito está de acordo com a estimativa do majorante do erro na interpolação por splines cúbicos dada pela equação (23). É muito claro que, na figura

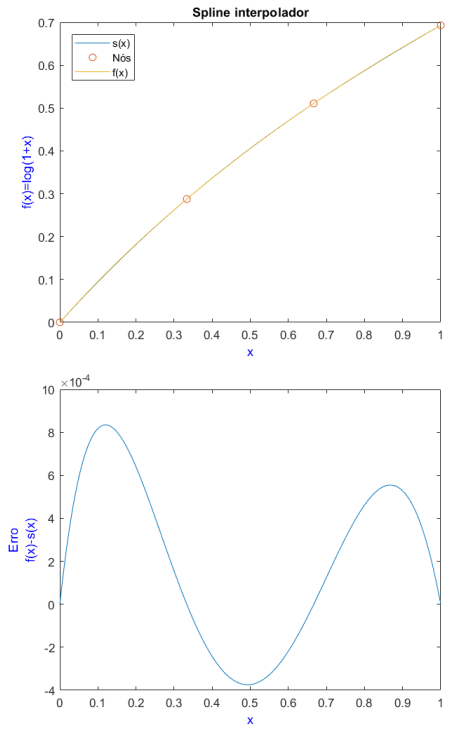

Fig. 1: Gráfico e erro absoluto da interpolação de $f(x) = \log(1+x)$ em $[0,1]$ com espaçamento uniforme e $n = 3$ troços ($h = 1/3$).

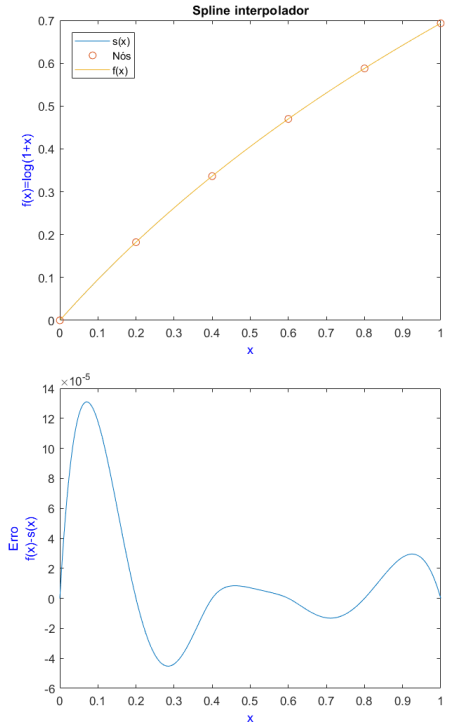

Fig. 2: Gráfico e erro absoluto da interpolação de $f(x) = \log(1+x)$ em $[0,1]$ com espaçamento uniforme e $n = 5$ troços ($h = 0.2$).

2 (e derivadas para as mesmas condições no anexo A), o erro

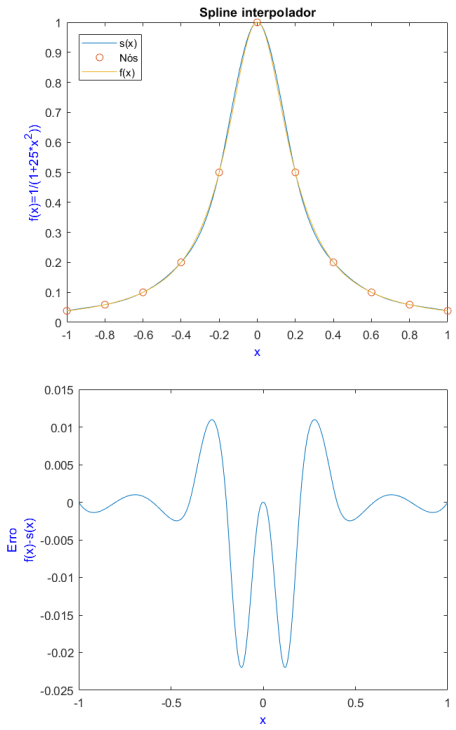

Fig. 3: Gráfico e erro absoluto da interpolação de $f(x) = \frac{1}{1+25x^2}$ *em* $[-1, 1]$ *com espaçamento uniforme e* $n = 10$ *troços (*$h = 0.2$*).*

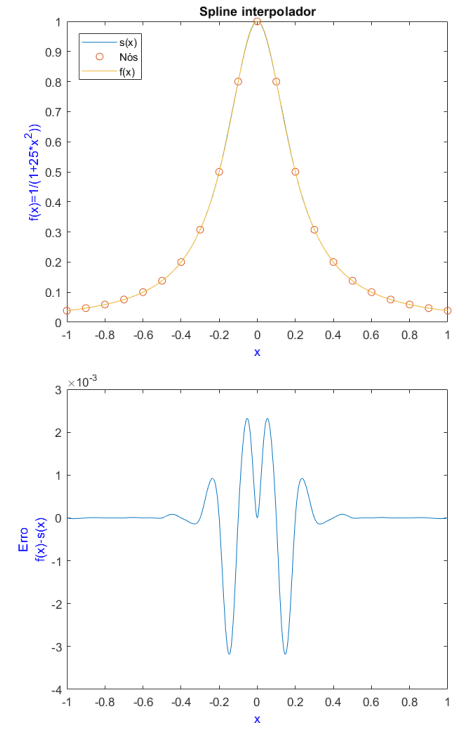

Fig. 4: Gráfico e erro absoluto da interpolação de $f(x) = \frac{1}{1+25x^2}$ *em* $[-1, 1]$ *com espaçamento uniforme e* $n = 20$ *troços (*$h = 0.1$*).*

é significativamente superior no primeiro troço, em que:

$$f^{(4)}(x) = -\frac{6}{(1+x)^4}$$

tem um valor bastante elevado, comparativamente com os restantes troços.

Reparamos também que no caso das figuras 3, 4 (e derivadas para as mesmas condições no anexo A) o erro associado é notoriamente mais alto nos troços compreendidos no intervalo $[-0.4, 0.4]$, em que

$$f^{(4)}(x) = -\frac{4500000x^2}{(25x^2+1)^4} + \frac{15000}{(25x^2+1)^3} + \frac{150000000x^4}{(25x^2+1)^5}$$

toma, também, valores significativamente maiores do que nos restantes troços.

Podemos também notar que nas figuras 2 e 3 para espaçamento igual ($h = 0.2$) temos um erro absoluto máximo duas ordens de grandeza superior na figura 3, em relação à figura 2, dado que $f^{(4)}(x)$ é significativamente menor nesta última figura. Este facto é também visível quando analisamos as figuras, para as mesmas condições, no anexo A.

É ainda importante salientar que, como podemos observar, o erro cometido na interpolação aumenta nas sucessivas derivadas, como seria de prever por (23). Para além disso, sendo construído por ramos de terceiro grau, a segunda derivada de um spline cúbico será uma reta em cada um dos troços. Por essa razão, como aliás se verifica no anexo A, estes não são adequados para extrapolar corretamente a segunda derivada de uma função. Em particular, como no spline Knot-a-Knot há continuidade nos troços 1 e $n-1$, então para três troços a terceira derivada do spline é constante, como verificamos na figura 6.

B. Estudo do erro máximo vs espaçamento entre nós

Pretendemos, nesta subsecção, estudar a relação entre o erro absoluto máximo cometido na interpolação da função e extrapolação das suas primeira e segunda derivadas em função do espaçamento uniforme usado, h, que deverá ser, aproximadamente, majorado por (23). Estando particularmente interessados na ordem de convergência do spline, ou seja, a potência do factor h em (23), estudaremos a relação entre $\log(||E||_\infty)$ e $\log(h)$ na expectativa de verificar a relação (25) e as equivalentes para as derivadas.

No anexo B estão representados os gráficos desta dependência para as funções $f_1(x)$ e $f_2(x)$, para o caso particular do spline Knot-a-Knot.

Como podemos observar a partir dos resultados da tabela I, nem todos os tipos de spline verificam a relação (23) consistentemente. No entanto, é em todos os casos verificada uma dependência aproximadamente linear, com a exceção da segunda derivada do spline natural. Verificamos, com facilidade, que a ordem de convergência dos splines completo e knot-a-not é, aproximadamente, a prevista por (23), sendo que a do spline completo é a que se encontra mais próxima, o que está de acordo com o deveríamos esperar de (29), não se tratando de uma aproximação para este tipo de spline. No entanto, reparamos que, no que diz respeito ao spline natural, o declive é consistentemente distinto do esperado para $f_i(x)$ com $i = 1, ..., 4$, mas próximo para $f_5(x)$, o que nos leva a concluir que (24) não modela corretamente o erro

Spline/Função	$f_1(x)$	$f_2(x)$	$f_3(x)$	$f_4(x)$	$f_5(x)$
Natural	2.0694	1.9603	2.0508	2.0777	4.0218
Completo	3.955	3.774	4.0086	3.9799	4.0357
Not-a-Knot	3.9276	3.4986	4.1213	4.3193	4.4305
Spline/Função	$f_1'(x)$	$f_2'(x)$	$f_3'(x)$	$f_4'(x)$	$f_5'(x)$
Natural	0.9961	0.9520	0.9817	1.0018	3.0221
Completo	2.9468	2.7891	3.0125	3.0314	3.0344
Not-a-Knot	2.8571	2.4460	3.0401	3.3157	3.5938
Spline/Função	$f_1''(x)$	$f_2''(x)$	$f_3''(x)$	$f_4''(x)$	$f_5''(x)$
Natural	0.0000	0.0000	0.0000	0.0000	1.9894
Completo	1.9483	1.6792	2.0018	1.9876	1.9832
Not-a-Knot	1.8686	1.4537	2.0387	2.2934	2.3337

TABLE I: Declive do gráfico de $\log(||E||_\infty)$ *em função de* $\log(h)$ *para a interpolação de vários tipos splines em* $[0,1]$ *com espaçamento uniforme e* $h \in [0.0013, 1/3]$ *para várias funções (e suas derivadas):* $f_1(x) = \log(1+x)$, $f_2(x) = \frac{1}{1+25x^2}$, $f_3(x) = \sin(x^2)$, $f_4(x) = \tanh(x)$ *e* $f_5(x) = sin(\pi x)$

absoluto na interpolação por um spline natural para qualquer tipo de funções. Vejamos que a função $f_5(x)$ é a única em que ambos os extremos do intervalo têm segunda derivada nula, a condição de fecho deste tipo de spline. Assim, seria de esperar, como aliás verificamos, que (24) seja satisfeita apenas para funções com estas características.

Em particular, para o caso da segunda derivada do spline natural, notamos que não existe uma dependência do erro máximo em h para as primeiras quatro funções. Este fenómeno deve-se ao facto de o erro máximo cometido corresponder, para qualquer valor de h ao erro cometido na fixação de segundas derivadas nulas nos nós extremos.

C. Aplicação prática - viga elástica

Pretendemos, nesta subsecção, estudar um modo de vibração de uma viga encastrada em ambas as extremidades. Para tal, consideremos $w(x)$, a função do deslocamento vertical da viga em função da distância a uma das suas pontas. Temos, do modelo de uma viga de Euler-Bernoulli, que, nestas condições,

$$w'(0) = w'(L) = 0 \tag{30}$$

em que L é o comprimento da viga.

Assim, se determinarmos, experimentalmente, o deslocamento vertical de um conjunto de pontos da viga, para um mesmo instante de tempo, de um modo de vibração, podemos tentar interpolar os pontos obtidos por forma a obter uma aproximação de $w(x)$. Dadas as condições (30), e, comparando-as com (12), é fácil verificar que o spline interpolador mais adequado é o completo, com derivadas nulas nos nós extremos. Desta forma, conseguimos interpolar os dados experimentais, resolvendo o sistema (10) por forma a obter as segundas derivadas em cada nó e determinar: $w(x)$, a rotação $w'(x)$ e curvatura $w''(x)$, através de (6), (7) e (2), respetivamente.

Para os dados experimentais no anexo C, obtivemos uma aproximação de $w(x)$, $w'(x)$ e $w''(x)$ representado nas figuras 19, 20 e 21, respetivamente, também no anexo C. A partir da análise dos gráficos concluímos que, de facto, esta interpolação é uma boa aproximação do verdadeiro comportamento da viga. No entanto, notamos que o gráfico da curvatura da viga (figura 21) é composto por segmentos de reta entre os pontos experimentais, quando na realidade deveria ser uma curva com continuidade nas suas derivadas. Este efeito é inevitável na utilização de splines cúbicos interpoladores, visto que a sua terceira derivada é descontínua de troço para troço.

V. Conclusão

Com este projeto, tivemos a oportunidade de estudar com algum detalhe os splines cúbicos, algumas das suas condições de fecho e de explorar algumas das suas propriedades.

Em primeiro lugar, concluímos que na sua generalidade a interpolação por splines cúbicos é uma ferramenta muito poderosa. Com efeito, tem propriedades que se revelam muito úteis para resolução de uma grande variedade de problemas, escolhendo as condições de fecho adequadas.

Em segundo lugar, desenvolvemos um estudo do erro cometido na interpolação por splines cúbicos como aproximação de uma função, e das derivadas do spline como aproximação das derivadas da função, com especial ênfase na ordem de convergência. Deste estudo concluímos que o spline completo é aquele cujo erro é mais fielmente modelado. A convergência do spline Not-a-Knot é também bastante próxima da estimativa explorada ao longo do projeto, como pudemos verificar. Já no caso do spline natural, verificamos que o erro induzido na interpolação é bem modelado apenas para funções cujas características justifiquem o uso deste tipo de spline, isto é, que sejam coerentes com as condições de fecho no intervalo de interesse.

Por último, munidos de uma equação geral para o erro, desenvolvemos um algoritmo simples de posicionamento adaptativo de nós, por forma a aproximar uma função com um erro inferior a um limite estabelecido. Com a aplicação deste método pudemos verificar as suas claras vantagens relativamente à interpolação com espaçamento uniforme. Sumariamente, essas vantagens englobam o uso de menos troços, a capacidade de interpolar uma função abaixo de um erro previamente estabelecido e uma distribuição mais uniforme do erro cometido.

References

[1] José Viriato Araújo dos Santos. *Problemas Splines Cúbicos*. Material de Estudo da unidade curricular de Matemática Computacional, 2018.

[2] Leonel Monteiro Fernandes. *Interpolação Polinomial*, volume 2. Material de Estudo da unidade curricular de Matemática Computacional.

[3] João Folgado. *Interpolação Polinomial*. Material de Estudo da unidade curricular de Matemática Computacional.

[4] Carl de Boor Garrett Birkhoff. *Error bounds for spline interpolation*, volume 13. Jounal of Mathematics and Mechanics, 1964.

[5] Antony Jameson. *Cubic Splines*. Department of Aeronautics and Astronautics, Stanford University.

[6] Gheorghe Micula. *A variational approach to spline functions theory*. General Mathematics, 2002.

[7] Irene A. Stegun Milton Abramowitz. *Handbook of Mathematical Functions with Formulas Graphs and Mathematical Tables*. 9 edition, 1970.

[8] Carl deBoor S.D. Conte. *Elementary Numerical Analysis*. McGraw-Hill, 1972.

[9] P. Sonneveld. *Errors in Cubic Spline Interpolation*, volume 3. Journal of Engineering Math, 1969.

VI. Anexo A

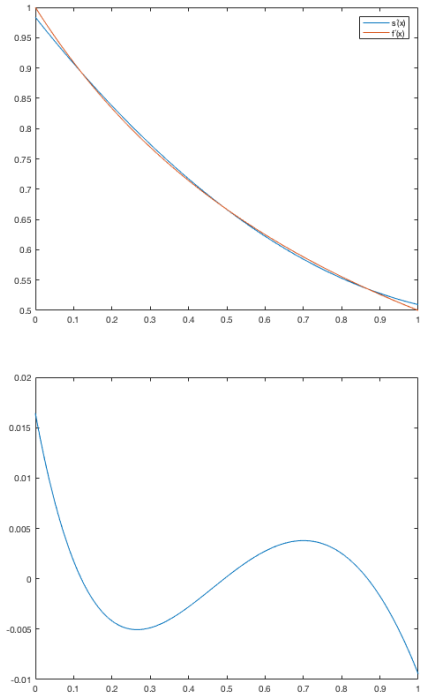

Fig. 5: Gráfico e erro absoluto da interpolação de $f'(x)$ com $f(x) = \log(1+x)$ em $[0,1]$ com espaçamento uniforme e $n = 3$ troços ($h = 1/3$).

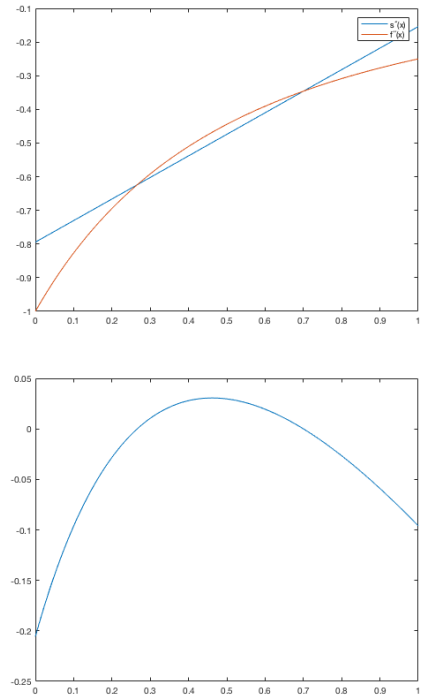

Fig. 6: Gráfico e erro absoluto da interpolação de $f''(x)$ com $f(x) = \log(1+x)$ em $[0,1]$ com espaçamento uniforme e $n = 3$ troços ($h = 1/3$).

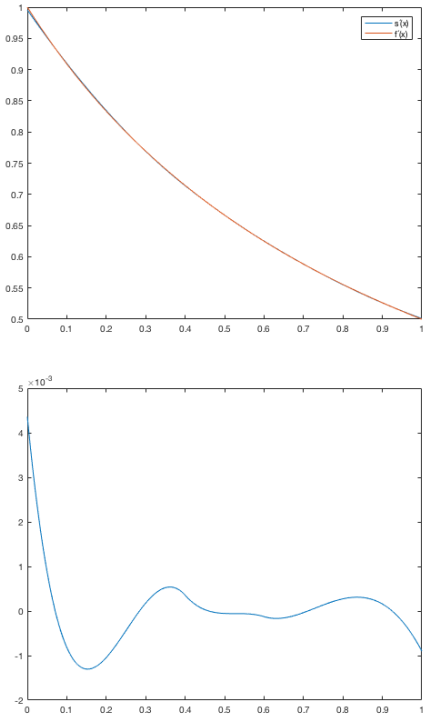

Fig. 7: Gráfico e erro absoluto da interpolação de $f'(x)$ com $f(x) = \log(1+x)$ em $[0,1]$ com espaçamento uniforme e $n = 5$ troços ($h = 0.2$).

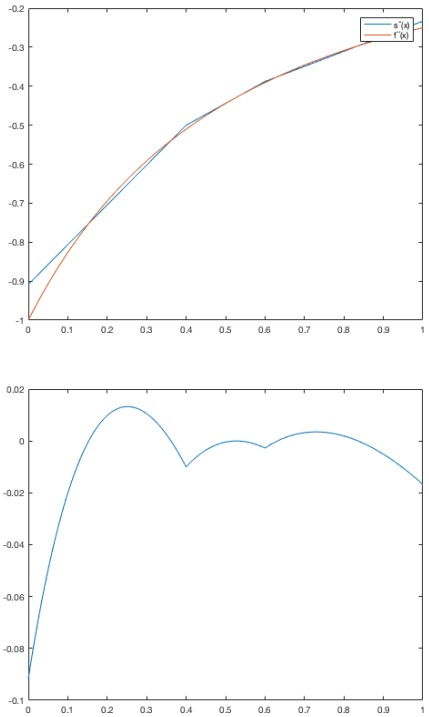

Fig. 8: Gráfico e erro absoluto da interpolação de $f''(x)$ com $f(x) = \log(1+x)$ em $[0,1]$ com espaçamento uniforme e $n = 5$ troços ($h = 0.2$).

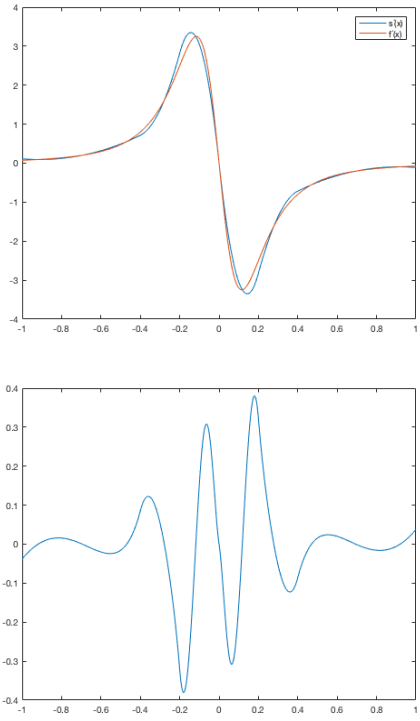

Fig. 9: Gráfico e erro absoluto da interpolação de $f'(x)$ com $f(x) = \frac{1}{1+25x^2}$ em $[-1, 1]$ com espaçamento uniforme e $n = 10$ troços ($h = 0.2$).

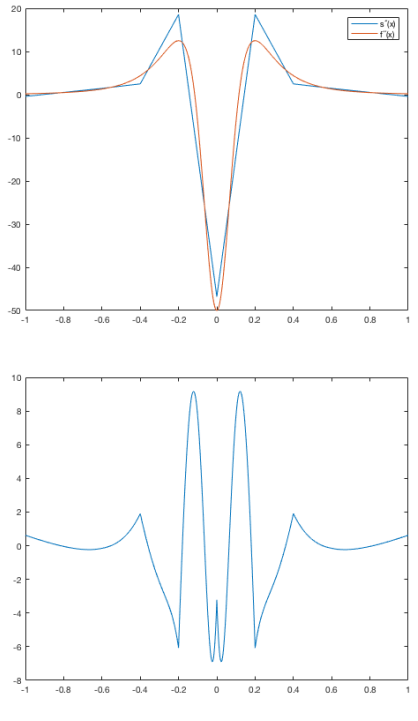

Fig. 10: Gráfico e erro absoluto da interpolação de $f''(x)$ com $f(x) = \frac{1}{1+25x^2}$ em $[-1, 1]$ com espaçamento uniforme e $n = 10$ troços ($h = 0.2$).

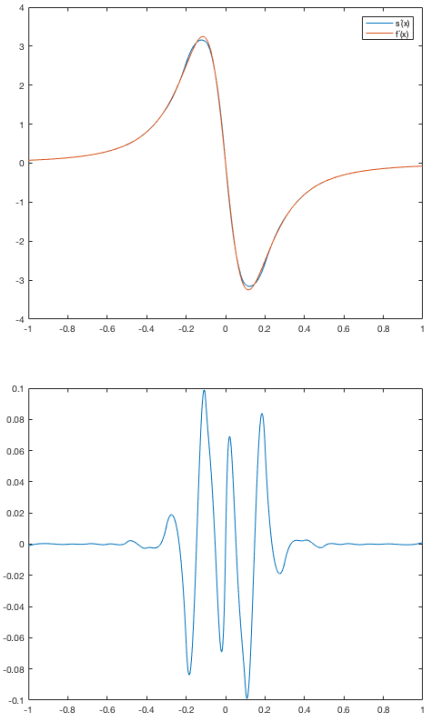

Fig. 11: Gráfico e erro absoluto da interpolação de $f'(x)$ com $f(x) = \frac{1}{1+25x^2}$ em $[-1, 1]$ com espaçamento uniforme e $n = 20$ troços ($h = 0.1$).

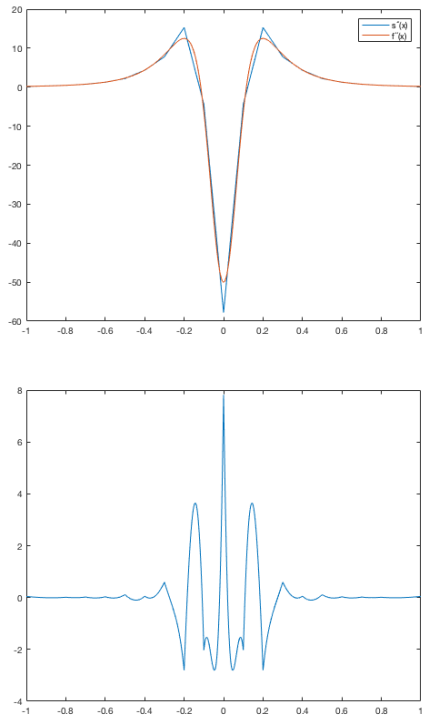

Fig. 12: Gráfico e erro absoluto da interpolação de $f'(x)$ com $f(x) = \frac{1}{1+25x^2}$ em $[-1, 1]$ com espaçamento uniforme e $n = 20$ troços ($h = 0.1$).

VII. Anexo B

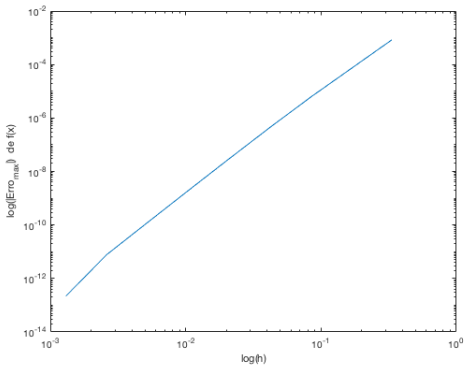

Fig. 13: Gráfico de $\log(||E||_\infty)$ *em função de* $\log(h)$ *para a interpolação por um spline not-a-knot de* $f(x) = log(1+x)$ *em* $[0,1]$ *com espaçamento uniforme,* $h \in [0.0013, 1/3]$ *e declive* 3.9276

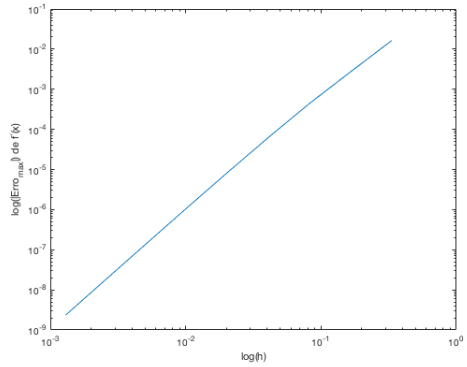

Fig. 14: Gráfico de $\log(||E||_\infty)$ *em função de* $\log(h)$ *para a aproximação por um spline not-a-knot de* $f'(x)$ *com* $f(x) = log(1+x)$ *em* $[0,1]$ *com espaçamento uniforme,* $h \in [0.0013, 1/3]$ *e declive* 2.8571

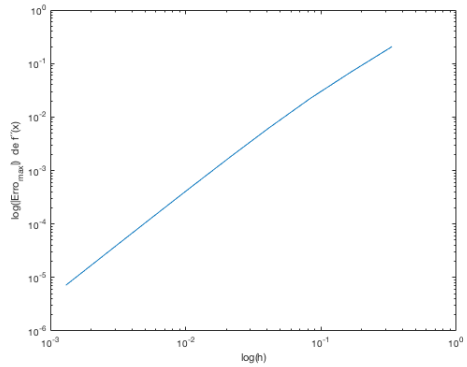

Fig. 15: Gráfico de $\log(||E||_\infty)$ *em função de* $\log(h)$ *para a aproximação por um spline not-a-knot de* $f''(x)$ *com* $f(x) = log(1+x)$ *em* $[0,1]$ *com espaçamento uniforme,* $h \in [0.0013, 1/3]$ *e declive* 1.8686

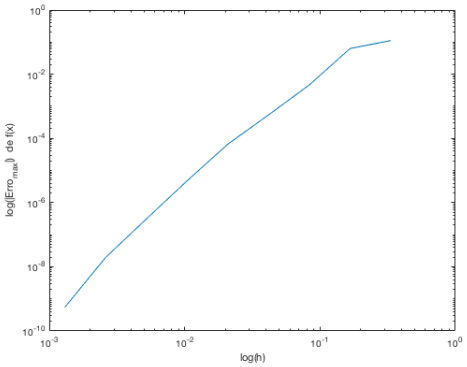

Fig. 16: Gráfico de $\log(||E||_\infty)$ *em função de* $\log(h)$ *para a interpolação por um spline not-a-knot de* $f(x) = \frac{1}{1+25x^2}$ *em* $[0,1]$ *com espaçamento uniforme,* $h \in [0.0013, 1/3]$ *e declive* 3.4986

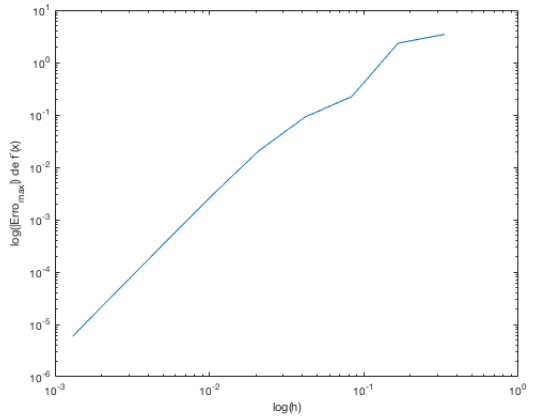

Fig. 17: Gráfico de $\log(||E||_\infty)$ *em função de* $\log(h)$ *para a aproximação por um spline not-a-knot de* $f'(x)$ *com* $f(x) = \frac{1}{1+25x^2}$ *em* $[0,1]$ *com espaçamento uniforme,* $h \in [0.0013, 1/3]$ *e declive* 2.4460

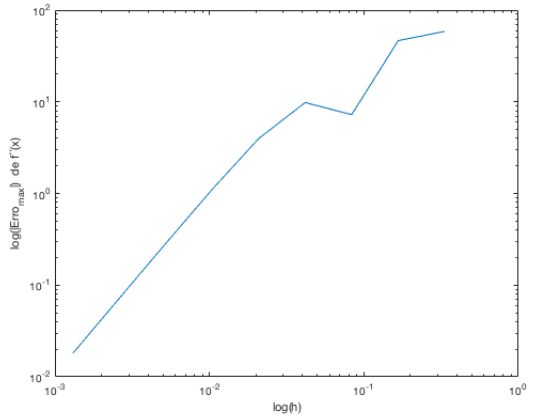

Fig. 18: Gráfico de $\log(||E||_\infty)$ *em função de* $\log(h)$ *para a aproximação por um spline not-a-knot de* $f'(x)$ *com* $f(x) = \frac{1}{1+25x^2}$ *em* $[0,1]$ *com espaçamento uniforme,* $h \in [0.0013, 1/3]$ *e declive* 1.4537

VIII. Anexo C

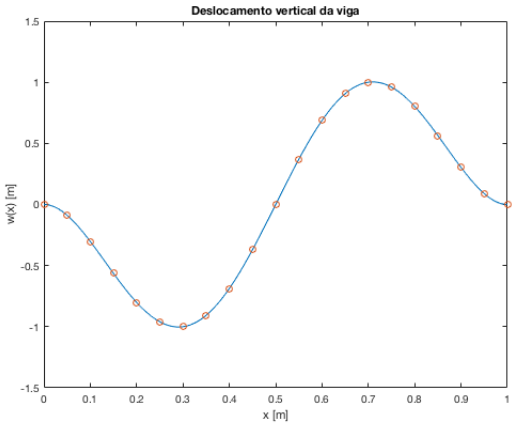

Fig. 19: Gráfico da interpolação dos dados experimentais do deslocamento vertical da viga, $w(x)$

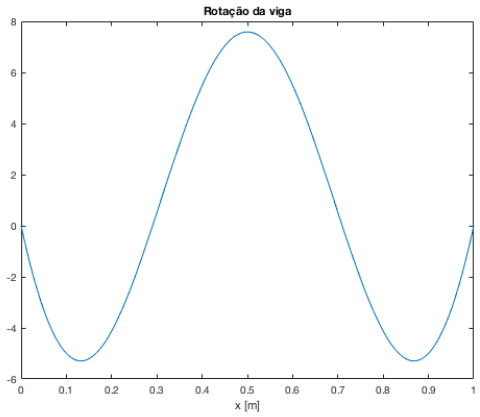

Fig. 20: Gráfico da rotação da viga, $w'(x)$

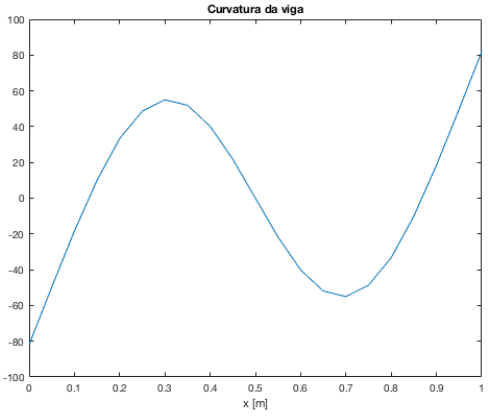

Fig. 21: Gráfico da curvatura da viga, $w''(x)$

x (m)	y
0.00	0.00000
0.05	-0.0890
0.10	-0.3027
0.15	-0.5636
0.20	-0.8016
0.25	-0.9597
0.30	-1.0000
0.35	-0.9067
0.40	-0.6872
0.45	-0.3701
0.50	0.00000
0.55	0.3701
0.60	0.6872
0.65	0.9067
0.70	1.0000
0.75	0.9597
0.80	0.8016
0.85	0.5636
0.90	0.3027
0.95	0.0890
1.00	0.0000

TABLE II: Dados experimentais utilizados para a aplicação de interpolação por splines cúbicos a um modo de vibração de uma viga

IX. Anexo D - Algoritmo adaptativo de interpolação

A. Introdução

Pretendemos, nesta secção, desenvolver um método adaptativo simples de interpolação por splines cúbicos.
Suponhamos que queremos interpolar $f(x) \in C^4$, em $[a, b]$, com um erro absoluto máximo de $\epsilon \in \mathbb{R}^+$. O objetivo do algoritmo é interpolar a função com um número mínimo de ramos, avaliando a função num número mínimo de pontos e de forma a que o erro absoluto seja inferior a ϵ, definido antecipadamente.
Dado que o spline completo é o que tem a propriedade da melhor aproximação, como analisado na secção II, e é, também, aquele para o qual é possível fazer um estudo do erro mais exato, como concluído da secção III, será o tipo de spline que usaremos.

B. Função de erro aproximada

Em primeiro lugar, é fulcral selecionar uma medida de erro adequada, como um critério para a colocação dos nós. Para isso, faremos uma aproximação para as relações estudadas na secção III, que se verifica, na prática, ser válida. Assim, seguidamente, exploraremos, qualitativamente, o porquê de esta ser uma boa aproximação.

Analisemos a equação (26) dada abaixo:

$$|E(f,x)| \leq \frac{(x-x_i)^2(x_{i+1}-x)^2}{24} f^{(4)}(\xi_i) + \frac{(x-x_i)(x_{i+1}-x)}{24} \frac{||h||^3}{h_{i+1}} ||f^{(4)}||_\infty \tag{31}$$

em que $||h|| = \max(h_i)$ e $\xi_i \in [x_i, x_{i+1}]$.
Verificamos que o erro depende, não só da quarta derivada da função no intervalo $[x_i, x_{i+1}]$, como também do majorante da derivada em $[x_0, x_n]$. E, analogamente, não depende apenas de h_{i+1} mas também de $||h||$.

De [9] temos que $|E(f,x)|$ tem contribuições de dois efeitos, representados pelos dois termos da expressão (31). Na verdade, o primeiro termo:

$$\frac{(x-x_i)^2(x_{i+1}-x)^2}{24} f^{(4)}(\xi_i) \tag{32}$$

é o majorante do erro cometido na interpolação por um spline de Hermite cúbico nos nós x_i com $i = 0, .., n$, que verificamos ser um erro com dependência local.
Um spline de Hermite é composto por troços de polinóminos de terceiro grau, com continuidade na função e na primeira derivada com as seguintes propriedades:

$$S(x_i) = f(x_i) \qquad S'(x_i) = f'(x_i) \tag{33}$$

E o segundo termo:

$$\frac{(x-x_i)(x_{i+1}-x)}{24} \frac{||h||^3}{h_{i+1}} ||f^{(4)}||_\infty \tag{34}$$

que tem em consideração o facto de as derivadas do spline nos nós não corresponderem às derivadas da função nesses mesmo nós, para uma distribuição genérica destes, o leva a que a expressão tenha uma dependência global.
Pretendemos, por forma a simplificar o algoritmo e não sobrestimar desnecessariamente o erro cometido, obter uma aproximação válida do erro que dependa, em primeira aproximação, apenas das derivada no troço considerado e do seu espaçamento, ou seja, que tenha uma dependência apenas local.
Pela forma como estamos a construir o algoritmo adaptativo, pretendemos aumentar a densidade de nós em zonas em que a quarta derivada da função tome valores mais elevados, pelo que estamos a refinar a partição usada nessas zonas. Assim, o erro que a interpolação em troços em que a quarta derivada é elevada induz nos restantes troços é significativamente reduzido. Desta forma, podemos considerar que, para $h_{i+1} = x_{i+1} - x_i$ pequeno, esse erro depende, aproximadamente, do majorante da quarta derivada apenas nos troços adjacentes. Para além disso, considerando h_{i+1} suficientemente pequeno, as derivadas nos troços adjacentes podem ser aproximadas pela derivada no troço i. E, por essa razão, o espaçamento h_{i+1}, pela a forma como estamos a construir o algoritmo, usado nos troços adjacentes será, também, idêntico, em primeira aproximação.

Assim, aproximamos (34) por:

$$\frac{(x-x_i)(x_{i+1}-x)}{24} \frac{h_{i+1}^3}{h_{i+1}} ||f^{(4)}||_{[x_i, x_{i+1}]} \tag{35}$$

É de notar que (35) é da mesma forma que (32), no sentido em que tem uma dependência local e depende de h_{i+1}^4.
Para além disso, majorando (31) usando o mesmo raciocínio do empregado na seccção III, temos a seguinte função de erro aproximada:

$$|\hat{E}_i| \leq \frac{5}{385} h_{i+1}^4 ||f^{(4)}||_{[x_i, x_{i+1}]} \tag{36}$$

É de extrema importância notar, contudo, que esta aproximação é apenas válida para espaçamentos suficientemente pequenos.

C. Estimativa da quarta derivada de $f(x)$

Para estimar o majorante da quarta derivada de $f(x)$ em $[x_i, x_{i+1}]$ utilizaremos diferenças finitas. Em particular, por forma a simplificar o algoritmo, calcularemos esta diferença finita com base em apenas cinco pontos da função. De [7] temos que:

$$f^{(4)}(a_i) = \frac{1}{h^{*4}} (f(a_0) - 4f(a_1) + 6f(a_2) - 4f(a_3) + f(a_4)) \tag{37}$$

para $i = 0, .., 4$, com a_i espaçados uniformemente e $a_{i+1} - a_i = h^*$.

Faz sentido que, para que a aproximação calculada na subsecção anterior tenha validade, tenhamos de conseguir estimar com alguma precisão $f^{(4)}(x)$ no troço i. Assim, para

cada troço i, avaliaremos a função para 3 pontos interiores espaçados uniformemente. Desta forma, o espaçamento entre estes pontos auxiliares no troço i, h_i^*, é dado por:

$$h_i^* = \frac{h_{i+1}}{5} \tag{38}$$

Temos, substituindo (38) em (31), que:

$$|\hat{E}_i| \leq \frac{625}{77}(f(a_0^i) - 4f(a_1^i) + 6f(a_2^i) - 4f(a_3^i) + f(a_4^i)) \tag{39}$$

em que a_j^i são os pontos auxiliares interiores no troço i, com $i = 1, .., n$ e $j = 0, ..., 4$.

D. Estimativa da segunda derivada nos nós extremos

Como estamos a utilizar um spline cúbico completo para a interpolação, é necessário aproximar, com precisão, a primeira derivada da função nos nós extremos. Temos, das subsecções anteriores, que, em cada troço, terão de ser avaliados três pontos interiores. Assim, temos, para cada troço extremo, um total de cinco pontos (o nó extremo, os três pontos interiores e o nó seguinte), com um espaçamento uniforme entre eles: h_1^* e h_n^*, respetivamente para o primeiro e último troços.
Assim, podemos empregar uma diferença finita progressiva e regressiva, respetivamente no primeiro e último nós, com os cinco pontos que temos disponíveis.
Temos de [7] que:

$$f'(a_0^1) = \frac{1}{24h_1^*}(-50f(a_0^1) + 96f(a_1^1) - 72f(a_2^1) + 32f(a_3^1) - 6f(a_4^1))$$

$$f'(a_4^n) = \frac{1}{24h_n^*}(6f(a_0^n) - 32f(a_1^n) + 72f(a_2^n) - 96f(a_3^n) + 50f(a_4^n))$$

E. Algoritmo do método

Podemos, assim, formalizar o algoritmo usado. Na verdade, a essência do algoritmo é calcular, para cada troço, o erro cometido na interpolação a partir da aproximação (39), sendo que foram já previamente calculados os pontos auxiliares interiores ao troço. Se essa aproximação for superior a ϵ, então dividimos o intervalo em dois de igual largura, procedendo, de seguida, à determinação dos pontos interiores nos novos troços, caso contrário prosseguimos para o cáculo de erro no próximo troço. É evidente que este processo iterativo terminará quando o erro aproximado em cada troço seja inferior a ϵ.
É de salientar que, para os primeiros passos da interação, as aproximações (39) e (37) não são válidas, pelo que a iteração é começada com um número inicial de troços. Verificamos que dois troços são suficientes, na grande generalidade dos casos, para uma boa aproximação, pelo que, no algoritmo implementado em MATLAB, este será o número inicial de troços. É de notar, para além disso, que tendo calculado valores auxiliares para o interior dos troços, esses valores não vão ser usados para interpolação, uma vez que um dos objetivos é simplificar a expressão final o mais possível.

F. Aplicação do método adaptativo e comparação o uniforme

Nesta subsecção, aplicaremos o método descrito nas subsecções anteriores e compararemos a sua eficácia em relação à interpolação por um spline completo, em que os nós têm espaçamento uniforme.
Verificamos, de facto, o bom funcionamento do algoritmo de interpolação adaptativo que desenvolvemos. Na verdade, reparamos que o erro real na interpolação foi inferior a ϵ com alguma margem. É de notar ainda que, pela comparação das figuras 22 com 23 e 24 com 25, verificamos que nos gráficos do erro absoluto no algoritmo adaptativo o erro é distribuído de forma bastante uniforme, enquanto que no caso da interpolação com espaçamento uniforme, é notória uma grande variação, dado que o erro em alguns troços é várias ordens de grandeza superior ao nos restantes. Percebemos, assim, que os objetivos para o posicionamento dos nós foram cumpridos, na medida em que a densidade de nós é a mínima possível e de forma a que simultaneamente o erro cometido seja inferior a ϵ.

É de notar ainda que quando comparamos ambos os algoritmos temos de ter em conta que os gráficos 23 e 25 foram selecionados, manualmente, de forma a que o erro real fosse inferior a 0.01, enquanto que no caso do algoritmo adaptativo a colocação dos nós foi feita pelo próprio algoritmo, com base no valor de $\epsilon = 0.01$.
Salientemos ainda que, dado o número mais reduzido de nós usados na interpolação adaptativa, a expressão da função interpoladora é mais simples neste caso quando comparada com a obtida para espaçamento uniforme.

G. Desvantagens e melhoramentos possíveis

Em primeiro lugar, temos que o erro real na interpolação é, em ambos os casos, uma ordem de grandeza inferior a ϵ. Assim o majorante do erro está, nestes exemplos, a sobrestimar demasiado o erro cometido. Um melhoramento possível consistiria em analisar com mais detalhe a expressão (31) e aproximar um majorante que permitisse, com menos nós, um erro inferior a ϵ.
Em segundo lugar, uma das grandes desvantagens deste método é o facto de em cada troço serem utilizados três nós interiores auxiliares, o que significa que a função é avaliada em $4n - 3$ pontos, em que n é o número total de troços, em que uma fração considerável não são posteriormente utilizados para a interpolação. No entanto, estes nós são necessários, neste algoritmo, para garantir que o erro é corretamente majorado.

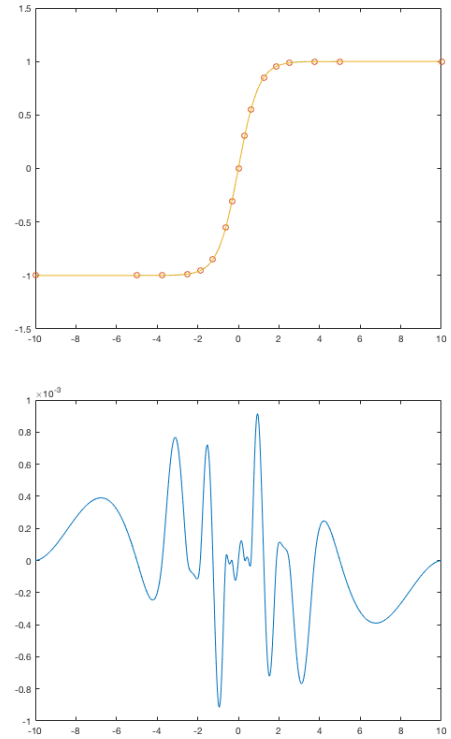

Fig. 22: Gráfico e erro absoluto da interpolação adaptativa de $f(x) = \tanh(x)$ *em* $[-10, 10]$ *com* $\epsilon = 0.01$ *e erro máximo cometido de* 0.000915

Fig. 23: Gráfico e erro absoluto da interpolação uniforme de $f(x) = \tanh(x)$ *em* $[-10, 10]$ *com* $n = 25$ *(*n *mínimo para erro inferior a* 0.01*) e erro máximo cometido de* 0.004577

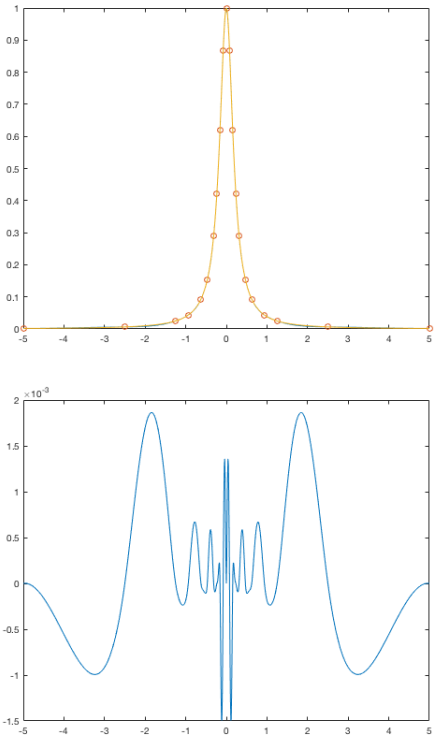

Fig. 24: Gráfico e erro absoluto da interpolação adaptativa de $f(x) = \frac{1}{1+25x^2}$ *em* $[-5, 5]$ *com* $\epsilon = 0.01$ *e erro máximo cometido de* 0.001863

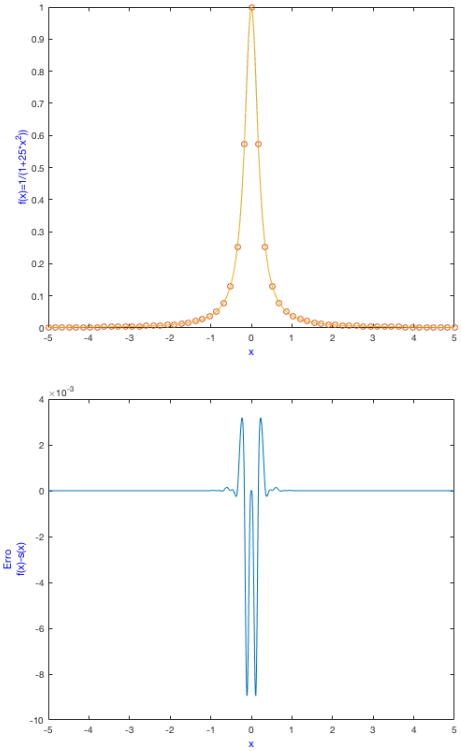

Fig. 25: Gráfico e erro absoluto da interpolação uniforme de $f(x) = \frac{1}{1+25x^2}$ *em* $[-5, 5]$ *com* $n = 58$ *(*n *mínimo para erro inferior a* 0.01*) e erro máximo cometido de* 0.008946

```
function Splines()
% É a função que deverá ser executada pelo utilizador
% Função encarregue de toda a interação com o utilizador.
% É a função que permite escolher a operação a efetura e os
% parametros desejados.

% Menu inicial
    fprintf("Bem vindo! ");
    fprintf("Selecione das alternativas abaixo a desejada:\n");
    fprintf("1 - Interpolação de uma função e suas derivadas\n");
    fprintf("2 - Estudar a ordem de convergência do erro\n");
    fprintf("3 - Interpolar um modo de vibração de uma viga\n");
    fprintf("4 - Interpolação adaptativa de uma função\n");
    fprintf("5 - Desenhar o spline guardado num ficheiro\n");
    fprintf("Escolha: ");

    % Ler a escolha e verificar a a sua validade
    escolha =str2double(input('','s'));
    while(isnan(escolha) || not(escolha == 1 || escolha ==2 || escolha
 ==3 || escolha == 4 || escolha ==5))
        fprintf('Opção inválida. Escolha:');
        escolha = str2double(input('','s'));
    end

    % Escolher a operação a executar consoanete a escolha do
 utilizador
    switch(escolha)
        case 1
            % Por forma a evitar usar calculo simbóloco apenas é
            % possível representar a primeira e segunda derivada do
            % spline sem análise do erro
            interpolarFuncao();
        case 2
            erro();
        case 3
            vigaVibracao();
        case 4
            adaptativo();
        case 5
          fprintf("Digite o nome do ficheiro para extrair os dados:
 ");
          filename = input('','s');
          plotSplineFromTxt(filename);
    end

end

function interpolarFuncao()
% Função que recolhe todos os parâmetros necessários para
% interpolar uma função e executa o comando
     d = selecinarFuncaoInteresse();
     k = selecionarFuncao(0);
```

```
    s = selecionarSpline();
    [a,b] = selecionarIntervalo();
    n = selecionarNtrocos(s);
    filenameS = saveSplineFileName();
    splineUniform(s-1,k,a,b,n,d,1,filenameS);
end

function erro()
% Função que recolhe todos os parâmetros necessários para fazer
% a análise do erro em função do espaçamento. Em particular,
% estudar os declive do gráfico log(Emax) vs log(h) por forma a
% estudar a ordem de convergência do spline escolhido
    k = selecionarFuncao(1);
    s = selecionarSpline();
    [a,b] = selecionarIntervalo();
    errorAnalysis(s-1,k,a,b);
end

function vigaVibracao()
% Função que pede o nome do ficheiro onde estão guardados os
% dados da viga e executa a interpolação desses dados.
    fprintf("Digite o nome do ficheiro para extrair os dados: ");
    filename = input('','s');
    filenameS = saveSplineFileName;
    beam(filename, filenameS);
end

function adaptativo()
    k = selecionarFuncao(0);
    [a,b] = selecionarIntervalo();
    epsl = limiteErro();
    splineAdaptive(k,a,b,epsl,1);
end

function k = selecionarFuncao(u)
% Função que permite ao utilizador selecionar uma função das já
% predefinidas ou introduzir uma nova função.
% Recebe u que para: u = 1 o utilizador não pode definir uma função
%                  : u = 0 o utilizador pode definir uma função
    fprintf("\nSelecione a função a interpolar:\n");
    fprintf("1 - log(1+x)\n");
    fprintf("2 - 1/(1+25*x^2)\n");
    fprintf("3 - sin(x^2)\n");
    fprintf("4 - tanh(x)\n");
    fprintf("5 - sin(pi*x)\n");
    if(u==0)
        fprintf("6 - Introduzir outra função\n");
    end
    fprintf("Escolha: ");

    k =str2double(input('','s')); %verificar aa validade da escolha
    while(isnan(k) || not(k == 1 || k ==2 || k ==3 || k == 4 || k == 5
 || k == 6-u))
        fprintf('Opção inválida. Escolha:');
```

```
        k = sscanf(input('','s'), "%d");
    end
    % se o utilizador escolher a função a variável k passa a ser
    % um 'function_handle' que a função funChoice consiguirá
    %intrepertar corretamente
    if(k == 6)
         fprintf("Introduzir função de variável x: ");
         k = str2func(['@(x)' input('','s')]);
    end
end

function s = selecionarSpline()
% Função que permite selecionar o tipo de spline interpolador

    fprintf("\nSelecione o tipo de spline a usar:\n");
    fprintf("1 - Natural\n");
    fprintf("2 - Completo\n");
    fprintf("3 - Not-a-Knot\n");
    fprintf("Escolha: ");

    s =sscanf(input('','s'),"%d");
    % verificar a avalidade da escolha
    while(isnan(s) || not(s == 1 || s ==2 || s ==3 || s == 4))
        fprintf('Opção inválida. Escolha:');
        s =sscanf(input('','s'),"%d");
    end
end

function [a,b] = selecionarIntervalo()
% Função que permite selecionar o intervalo no qual será feita a
% interpolação
    fprintf("\nÉ imperativo que a função escolhida esteja definida no
 intervalo selecionado.\n");
    fprintf("Defina o intervalo a interpolar na forma a,b: ");

    while true % verificar a avalidade do intervalo
        A = sscanf(input('','s'),"%f,%f");
        % verificar que a amplitude do intervalo é superior a 0.03
        if(max(size(A)) == 2 && A(2)> A(1) + 0.03)
            break;
        end
        fprintf("Inválido. Intervalo na forma a,b: ");
    end
    a = A(1);
    b = A(2);
end

function n = selecionarNtrocos(s)
% Função que permite selecionar o número de troços a usar na
% interpolação
    fprintf("Número de troços: ");
     while true %verificara a validade ad escolha
        n = sscanf(input('','s'),"%d");
        if(max(size(n)) == 1 && n>0 && not(n<3 && s == 3))
```

```
            break;
          elseif(n<3 && s == 3)
            % fprintf em comandos separados por uma questao de
            % formataçãodo código
            fprintf("O spline Not-a-Knot necessita de no mínimo 3
 troços.");
            fprinf(" Número de troços: ");
          else
            fprintf("Inválido. Intervalo na forma a,b: ");
          end

      end
end

function d = selecinarFuncaoInteresse()
% Função que permite selecionara os gráficos que predende estudar
    fprintf("\nSe decidir estudar derivadas da função poderá,
 apenas, comparar o valor obtido com o real se optar por uma fução
 predefinida");
    fprintf("\nSelecione o que pretende estudar:\n");
    fprintf("1 - Função\n");
    fprintf("2 - Função e sua 1ª derivada\n");
    fprintf("3 - Função, 1ª e 2ª derivadas\n");
    fprintf("Escolha: ");

    s =sscanf(input('','s'),"%d");
    % verificar a a validade da escolha
    while(isnan(s) || not(s == 1 || s ==2 || s ==3 ))
        fprintf('Opção inválida. Escolha:');
        s =sscanf(input('','s'),"%d");
    end

    if(s == 1)
        d = [1 0 0];
    elseif(s == 2)
        d = [1 1 0];
    else
        d = [1 1 1];
    end
end

function filenameS = saveSplineFileName()
%Função que permite ao utilizador introduzir o nome do ficheiro em que
%guardará os dados do spline interpolador
    % fprintf em comandos separados por uma questao de formatação
    % do código
    fprintf("Se não pretender gurdar o spline carregue em ENTER, caso
 ");
    fprintf("contrário digite o nome do ficheiro em que pretende
 gurdar: ");
    filenameS = input('','s');
end
```

```
function epsl = limiteErro()
    while true %verificara a validade do erro
        fprintf("Introduza o limite do erro desejado: ");
        epsl = sscanf(input('','s'),"%f");
        if(epsl>10*eps)
            break;
        end
        fprintf("Inválido. Limite do erro: ");
    end
end
```

Published with MATLAB® R2017b

```
function E = splineUniform(s,k,a,b,n,d,PlotGraph,filenameS)
% Descrição
% A função splineUniform interpola uma função dada pela função
% splineFunction através de um spline:
%                       - Natural para s=0
%                       - Completo para s=1
%                       - Not a Knot para s=2
% A função recebe como input:
%   -s: o tipo de spline a usar
%   -k: a função a interpolar
%   -a,b: os extremos do intervalo a interpolar
%   -n: número de troços a usar na interpolação
%   -d: vetor 1x3 que contem a informação da função ou derivadas
% de interesse
%   i.e. d(1)=1 significa fazer o estudo e gráfico para a
% função e 0 o oposto
%        d(2)=1 significa fazer o estudo e gráfico para a
% primeira derivada e 0 o oposto etc
%   -PlotGraph:
%       -0: Não fazer o plot da função interpolada, do spline
% interpolador nem dos nós usados para a interpolação
%       -1: Apesentar os resultados em um gráfico
% Esta função retorna o valor absolto do maior erro cometido na
% interpolação, E. E vai ser um vetor de dimensão igual ao número
% de 1s em d

% Funções Usadas e intervalos normalmente usados
% k -> fk(x)
% f1(x) = log(1+x)      em [0,1]
% f2(x) = 1/(1+25*x^2)  em [-1,1]
% f3(x) = sin(x^2)      em [0, 3*pi/2]
% f4(x) = tanh(x)       em [-20,20]

% Constantes
% número de pontos usados para o tracar o gráfico do spline
    nG = 1000;

% Iniciar nós
    h = (b-a)/n; % calcular o espaçamento (uniforme)
    x = [a:h:b]; % instanciar o vetor com os x dos n+1 nós
    % instanciar o valores de y correspondes a cada um dos nós
    y = funChoice(x,k,0);

% Obter o vetor M
% O vetor M, o vetor com o valor das segundas derivadas do spline
%interpolador em cada um dos nós, é obtido de forma diferente
% consoante o tipo de spline que usamos para interpolar a função k

    % Resolver o sistema para cada tipo de spline
    switch s
        case 0 % spline natural
            M = splineNatural(n,h,y);
```

```
        case 1 %spline completo
            % obter as derivadas da função interpolada nos nós
 extremos
            D = funChoice(x,k,1);
            M = splineClamped(n,h,D,y);
        case 2 % spline not a knot
            M = splineNotAKnot(n,h,y);
    end

% Criar gráfico da função, suas derivadas, e erros

    xG = [a:(b-a)/nG:b]; % instanciar o vetor com os x dos pontos
    % do gráfico calcular os valores do spline interpolador nos
    % pontos x onde pretendemos traçar o gráfico a partir da função
    % splineFunction

    E = [];

    if (d(1) == 1)
        yGs = splineFunction(xG,x,y,M);
        % calcular os valores da função interpolada para os pontos
        % x do gráfico
        yGf = funChoice(xG,k,0);
        % vetor do erro de cada ponto calculado para o gráfico
        yGe = yGf-yGs;
        E = [E max(abs(yGe))];
    end

    if (d(2) == 1)
        yGsD = splineFunctionD(xG,x,y,M);
        % calcular os valores da derivada da função para os pontos
        %x do gráfico
        if(~isa(k, 'function_handle'))
            yGfD = funChoice(xG,k,2);
            % vetor do erro de cada ponto calculado para o gráfico
            yGeD = yGfD-yGsD;
            E = [E max(abs(yGeD))];
        end
    end

    if (d(3) == 1)
        yGsDD = splineFunctionDD(xG,x,M);
        if(~isa(k, 'function_handle'))
            % calcular os valores da 2ª derivada da função para os
            % pontos x do gráfico
            yGfDD = funChoice(xG,k,3);
            % vetor do erro de cada ponto calculado para o gráfico
            yGeDD = yGfDD-yGsDD;
            E = [E max(abs(yGeDD))];
        end
    end


    % Gráficos das funções a ser estudadas
```

```
    if(PlotGraph)
        if(d(1) == 1)
            figure;
            subplot(2,1,1);
            plot(xG,yGf);
            hold on;
            scatter(x,y);
            plot(xG,yGs);
            title('f(x)')
            legend({'f(x)','Nós','s(x)'})
            subplot(2,1,2);
            plot(xG,yGe);
            title('Erro')
            hold off;
            fprintf("Erro máximo absoluto para h = %f uniforme para
f(x) é %f\n", h, E(1));
        end
        if(d(2) == 1)
            figure;
            if(~isa(k, 'function_handle'))
                subplot(2,1,2);
                plot(xG,yGeD);
                title('Erro');
                subplot(2,1,1);
                plot(xG,yGfD);
                hold on;
                fprintf("Erro máximo absoluto para h = %f uniforme
para f´(x) é %f\n", h,E(2));
                plot(xG,yGsD);
                title('f´(x)');
                legend({'f´(x)','s´(x)'})
            else
                plot(xG,yGsD);
                title('f´(x)');
            end

            hold off;

        end
        if(d(3) == 1)
            figure;
            if(~isa(k, 'function_handle'))
                subplot(2,1,2);
                plot(xG,yGeDD);
                title('Erro');
                subplot(2,1,1);
                plot(xG,yGfDD);
                hold on;
                fprintf("Erro máximo absoluto para h = %f uniforme
para f´´(x) é %f\n", h, max(abs(yGe)));
                plot(xG,yGsDD);
                title('f´´(x)');
                legend({'f´´(x)','s´´(x)'})
            else
```

```
                plot(xG,yGsDD);
                title('f´´(x)');
                hold off;
            end
        end
    end

    % Guardar, se pedido, o ficheiro com informação reltiva à função
    % interpoladora
    if (isa(filenameS, 'string') || isa(filenameS, 'char')...
                                    && ~strcmp(filenameS,""))
        saveSpline(x,y,M,filenameS);
    end

end
```

Published with MATLAB® R2017b

```
function M = splineNatural(n,h,y)
% Descrição
% A função splineNatural resolve o sistema de eqauções lineares
% necessário para determinar o valor da sengunda derivada de cada
% um dos nós do spline natural interpolador, com espaçamento uniforme.
% A função recebe como input:
%   -n: número de troços do polinómio interpolador
%   -h: espaçamento (uniforme) usado na interpolação
%   -y: o vetor dos valores da função interpolada em cada nó
% Retorna o vetor M das segundas derivadas em cada um dos nós

% Criar matriz spline natural
% Podemos reduzir o problema do spline natural à resolução de um
% sistema de equações lineares representado por uma matriz
% tridiagonal de dimensões n-1 x n-1 uma vez que M0 = 0 e Mn = n
% (cf. relatório)

    % Vetor dos n+1 valores das segundas derivadas em cada nó
    M = zeros(n+1,1);
    % criar a matriz A tridiagonal e B correspondente ao
    % sistema linear (AM=B) que pretendemos resolver (cf. relatório)
    %A = 4*diag(ones(n-1,1),0)+diag(ones(n-2,1),1)+...
                                        %diag(ones(n-2,1),-1);
    B = zeros(n-1,1);
    % introduzir os valores de B a partir do espamento, h, e vetor y
    % (cf. relatório)
    for i = 1:n-1
        B(i) = (6/h^2)*(y(i+2)-2*y(i+1)+y(i));
    end

    % As diagonais têm de ser passadas com a mesma dimensão,
    % no entanto, a(1) e c(end), os valores que ficariam
    % indefinidos não são usados pelo agoritmo
    M(2:n) = Thomas(ones(n-1,1),4*ones(n-1,1),ones(n-1,1),B);


end
```

Published with MATLAB® R2017b

```
function M = splineClamped(n,h,D,y)
% Descrição
% A função splineClamped resolve o sistema de eqauções lineares
% necessário para determinar o valor da sengunda derivada de cada
% um dos nós do spline completo interpolador, com espaçamento
% uniforme.
% A função recebe como input:
%   -n: número de troços do polinómio interpolador
%   -h: espaçamento (uniforme) usado na interpolação
%   -D: vetor (2x1) que corresponde às derivadas da função interpolada
% nos nós extremos
%   -y: o vetor dos valores da função interpolada em cada nó
% Retorna o vetor M das segundas derivadas em cada um dos nós

% Criar matriz spline completo
% Podemos reduzir o problema do spline completo à resolução de um
% sistema de equações lineares representado por uma matriz tridiagonal
% de dimensões n+1 x n+1 (cf. relatório)

    % criar a matriz A tridiagonal e B correspondente ao sistema
    % linear (AM=B) que pretendemos resolver (cf. relatório)
    % A tem a seguinte representação, no entanto, consideremos apenas
    %as suas digonais por forma a poupar memória
    %A = 4*diag(ones(n+1,1),0)+diag(ones(n,1),1)+diag(ones(n,1),-1);
    %A(1,1:2) = [2 1];
    %A(n+1,n:n+1) = [1 2];


    % introduzir os valores de B a partir do espamento, h, e vetor y
    %(cf. relatório)
    B = zeros(n+1,1);
    B(1) = (6/h)*((y(2)-y(1))/h-D(1));
    for i = 1:n-1
        B(i+1) = (6/h^2)*(y(i+2)-2*y(i+1)+y(i));
    end
    B(n+1) = (6/h)*(D(2)-(y(n+1)-y(n))/h);

    %M = A\B;
    % resolver o sistema de equações lineares
    d = 4*ones(n+1,1);
    d(1) = 2;
    d(end) = 2;
    % As diagonais têm de ser passadas com a mesma dimensão,
    % no entanto, a(1) e c(end), os valores que ficariam indefinidos
    % não são usados pelo agoritmo
    M = Thomas(ones(n+1,1), d, ones(n+1,1), B);


end
```

Published with MATLAB® R2017b

```
function M = splineNotAKnot(n,h,y)
% Descrição
% A função splineNotAKnot resolve o sistema de eqauções
% lineares necessário para determinar o valor da sengunda derivada
% de cada um dos nós do spline not a knot interpolador (i.e. com
% continuidade na terceira derivada nos nós 1 e n-1), com
% espaçamento uniforme.
% A função recebe como input:
%   -n: número de troços do polinómio interpolador
%   -h: espaçamento (uniforme) usado na interpolação
%   -y: o vetor dos valores da função interpolada em cada nó
% Retorna o vetor M das segundas derivadas em cada um dos nós

% Criar matriz spline not a knot
% Podemos reduzir o problema do spline not a knot à resolução
% de um sistema de equações lineares representado por uma matriz
% de dimensões n+1 x n+1 (cf. relatório)

    % criar a matriz A e B correspondente ao sistema de
    % equações lineares (AM=B) que pretendemos resolver
    % (cf. relatório)
    A = 4*diag(ones(n+1,1),0)+diag(ones(n,1),1)+diag(ones(n,1),-1);
    A(1,1:3) = [-1 2 -1];
    A(n+1,n-1:n+1) = [-1 2 -1];
    B = zeros(n+1,1);
    % introduzir os valores de B a partir do espamento, h, e vetor y
    % (cf. relatório)
    for i = 1:n-1
        B(i+1) = (6/h^2)*(y(i+2)-2*y(i+1)+y(i));
    end
    % resolver o sistema de equações lineares
    M = A\B;

end
```

Published with MATLAB® R2017b

```
function y = funChoice(x,k,n)
% Descrição
% A função funChoice é a função que é usada pelas funções:
 SplineNatural,
% SplineNotAKnot e SplineClamped para obter os valores da função
% interpolada no nós escolhidos e nos pontos selecionados para traçar
% o gráfico. É também usada para obter o valor da primeira e segunda
% derivadas
% No caso do SplineClamped é também utilizada para fornecer o
% valor das derivadas nos nós 0 e n.
% A função recebe como argumentos:
%   - x: um vetor de nós para o quais vão ser calculados o valor da
% função
%   - k: seleção da função a interpolar
%   - n: inteiro que seleciona a informação a fornecer acerca da
% função
%       - n=0: retorna um vetor com valor da função em cada ponto de x
%       - n=1: retorna um vetor (2x1) com o valor da 1ª derivada nos
% nós extremos
%       - n=2: retorna um vetor com valor de f'_k em cada ponto de x
%       - n=3: retorna um vetor com valor de f''_k em cada ponto de x
% É de notar que k pode ser um inteiro ou um 'function_handle'
% consoante seja uma função predefinida ou defina pelo utilizador,
% respetivamente.

% Funções Usadas e intervalos normalmente usados
% k -> fk(x)
% f1(x) = log(1+x)      em [0,1]
% f2(x) = 1/(1+25*x^2)  em [-1,1]
% f3(x) = sin(x^2)      em [0, 3*pi/2]
% f4(x) = tanh(x)       em [-20,20]

% Cálculo

if(~isa(k, 'function_handle')) %se a função for predefinida
    if n == 0 % valores da função em x
        switch k
            case 1
                y = log(1+x);
            case 2
                y = 1./(1+25*x.^2);
            case 3
                y = sin(x.^2);
            case 4
                y = tanh(x);
            case 5
                y = sin(pi*x);
            otherwise
                y = NaN(size(x));
        end
    elseif n == 1 % valores da derivada em x(0) e x(end)
        % instanciar o vetor da derivada da função k nos nós extremos
```

```
        y = zeros(2,1);
        switch k
            case 1
                y(1) = 1/(1+x(1));
                y(2) = 1/(1+x(end));
            case 2
                y(1) = -50*x(1)/((1+25*x(1)^2)^2);
                y(2) = -50*x(end)/((1+25*x(end)^2)^2);
            case 3
                y(1) = 2*x(1)*cos(x(1)^2);
                y(2) = 2*x(end)*cos(x(end)^2);
            case 4
                y(1) = 1-tanh(x(1))^2;
                y(2) = 1-tanh(x(end))^2;
            case 5
                y(1) = pi*cos(pi*x(1));
                y(2) = pi*cos(pi*x(end));
            otherwise
                y = NaN(2,1);
        end
    elseif n == 2
        switch k
            case 1
                y = 1./(1 + x);
            case 2
                y = -((50*x)./(1 + 25*x.^2).^2);
            case 3
                y = 2*x.*cos(x.^2);
            case 4
                y = 2./(1+cosh(2*x));
            case 5
                y = pi*cos(pi*x);
            otherwise
                y = NaN(size(x));
        end

    elseif n ==3

        switch k
            case 1
                y = -1./((1 + x).^2);
            case 2
                y = (5000*x.^2)./(1 + 25*x.^2).^3 -...
                                        50./(1 + 25.*x.^2).^2;
            case 3
                y = 2*cos(x.^2)-4*x.^2.*sin(x.^2);
            case 4
                y = -((8*sinh(x))./(3*cosh(x)+cosh(3*x)));
            case 5
                y = -pi*pi*sin(pi*x);
            otherwise
                y = NaN(size(x));
        end
```

```
    else
        y = NaN;
    end
else % se a função foi definida pelo utilizador
    if n == 0
        y = zeros(size(x));
        for i = 1:size(x,2)
           y(i) = k(x(i));
        end
    else
    % Se a função não for nenhuma das predefinidadas teriamos de usar
    % cálculo simbólico para determinar o valor da derivada no nós
    % extremos. Assim, decidimos aproximar a derivada através de uma
    % diferença finita com h = 0.00001
        y = zeros(1,2);
        h = 0.00001;
        y(1) = (-3*k(x(1))+4*k(x(1)+h)-k(x(1)+2*h))/((2*h));
        y(2) = (3*k(x(end))-4*k(x(end)-h)+k(x(end)-2*h))/(2*h);

    end
end
end
```

Published with MATLAB® R2017b

```
function yG = splineFunction(xG,x,y,M)
% A função splineFunction é a que com base nos pares (x,y) e
% nas segundas derivadas do spline interpolador anteriormente
% calculadas retorna o valor do spline em pontos entre os nós.
% Recebe como argumentos:
%   - xG: os pontos x em que queremos determinar o valor que o
% spline interpolador toma
%   - x: o vetor de nós de interpolação
%   - y: o vetor de valores da função interpolada em cada um dos nós
%   - M: o vetor dos valores da segunda derivada do spline
% interpolador em cada um dos nós
%   - a: o extremo inferior do intervalo onde estamos a interpolar a
%   a função
%   - h: o espaçamento (uniforme) usado na interpolação
% Retorna um vetor da mesma dimensão de xG que corresponde ao valor
% que spline interpolador toma para cada entrada de xG.
% Gerar pontos

    yG = zeros(size(xG)); % instanciar o vetor yG
    % varrer as entradas de xG e para cada valor calcular o valor
    %que o spline interpolador toma

    for j = 1:size(xG,2)
       % calcular o número do troço a que a entrada atual de
       % xG pertence.
       h = x(2)-x(1);
       i = ceil((xG(j)-x(1))/h);
       % no caso particular de a entrada xG se tratar do extremo
       % inferior temos de corrigir o valor calculado para o número
       % do troço
       if(i == 0)
           i = 1;
       end

       % Para valores muito reduzidos de h, os erros de
       % arrendondamento fazem i ser ligeiramente superior ao
       % valor inteiro correto, pelo que a função ceil, retorne
       % um valor errado
       if(i > size(x,2)-1)
           i = size(x,2)-1;
       end

       % Calcular o valor que o spline toma (cf. relatório)
       yG(j) = M(i)*((x(i+1)-xG(j))^3)/(6*h) ...
               +  M(i+1)*((xG(j)-x(i))^3)/(6*h)...
               + (y(i)-M(i)*((h^2)/6))*((x(i+1)-xG(j))/h)...
               + (y(i+1)-M(i+1)*((h^2)/6))*((xG(j)-x(i))/h);
    end
end
```

Published with MATLAB® R2017b

```
function yG = splineFunctionAdaptive(xG,x,y,M)

% A função splineFunction é a que com base nos pares (x,y) e
% nas segundas derivadas do spline interpolador anteriormente
% calculadas retorna o valor do spline em pontos entre os nós.
% Recebe como argumentos:
%   - xG: os pontos x em que queremos determinar o valor que
% o spline interpolador toma
%   - x: o vetor de nós de interpolação
%   - y: o vetor de valores da função interpolada em cada um
% dos nós
%   - M: o vetor dos valores da segunda derivada do spline
% interpolador em cada um dos nós
%   - a: o extremo inferior do intervalo onde estamos a interpolar
% a função
%   - h: o espaçamento (uniforme) usado na interpolação
% Retorna um vetor da mesma dimensão de xG que corresponde
% ao valor que spline interpolador toma para cada entrada de xG.

% Gerar pontos

    yG = zeros(size(xG)); % instanciar o vetor yG
    % varrer as entradas de xG e para cada valor calcular o valor
    % que o spline interpolador toma
    for j = 1:size(xG,2)
  % calcular o número do troço a que a entrada atual de xG pertence.

       i = 1;
       while i<size(x,2)-1
          if(xG(j)<=x(i+1))
             break
          end
          i = i+1;
       end

       % Calcular o valor que o spline toma (cf. relatório)
       h = (x(i+1)-x(i));
       yG(j) = M(i)*((x(i+1)-xG(j))^3)/(6*h)...
                + M(i+1)*((xG(j)-x(i))^3)/(6*h)...
                + (y(i)-M(i)*((h^2)/6))*((x(i+1)-xG(j))/h)...
                + (y(i+1)-M(i+1)*((h^2)/6))*((xG(j)-x(i))/h);
    end

end
```

Published with MATLAB® R2017b

```
function DyG = splineFunctionD(xG,x,y,M)
% A função splineFunction é a que com base nos pares (x,y)
% e nas segundas derivadas do spline interpolador anteriormente
% calculadas retorna o valor da derivada spline em pontos
% entre os nós.
% Recebe como argumentos:
%   - xG: os pontos x em que queremos determinar o valor que o
% spline interpolador toma
%   - x: o vetor de nós de interpolação
%   - y: o vetor de valores da função interpolada em cada um
% dos nós
%   - M: o vetor dos valores da segunda derivada do spline
% interpolador em cada um dos nós
% Retorna um vetor da mesma dimensão de xG que corresponde
% ao valor que a primeira derivada do spline interpolador toma
% para cada entrada de xG. É de notar que o espaçamento entre nós
% pode não ser uniforme
% Gerar pontos da primeira derivada

    DyG = zeros(size(xG)); % instanciar o vetor yG
    % varrer as entradas de xG e para cada valor calcular o
    % valor que o spline interpolador toma
    for j = 1:size(xG,2)
 % calcular o número do troço a que a entrada atual de xG pertence.
      i = 1;
       while i<size(x,2)-1
          if(xG(j)<=x(i+1))
             break
          end
          i = i+1;
       end

       h = (x(i+1)-x(i));

       % Calcular o valor que a derivada do spline toma
       % (cf. relatório)
       DyG(j) = - M(i)*((x(i+1)-xG(j))^2)/(2*h)...
                + M(i+1)*((xG(j)-x(i))^2)/(2*h)...
               + (y(i+1)-y(i))/h - ((M(i+1)-M(i))*h)/6;
    end

end
```

Published with MATLAB® R2017b

```
function DDyG = splineFunctionDD(xG,x,M)
% A função splineFunction é a que com base nos pares (x,y)
% e nas segundas derivadas do spline interpolador anteriormente
% calculadas retorna o valorda segunda derivada spline em pontos
% entre os nós.
% Recebe como argumentos:
%   - xG: os pontos x em que queremos determinar o valor que o
% spline interpolador toma
%   - x: o vetor de nós de interpolação
%   - y: o vetor de valores da função interpolada em cada um
% dos nós
%   - M: o vetor dos valores da segunda derivada do spline
% interpolador em cada um dos nós
% Retorna um vetor da mesma dimensão de xG que corresponde
% ao valor que a segunda derivada do spline interpolador toma para
% cada entrada de xG.

% Gerar pontos da segunda derivada

    DDyG = zeros(size(xG)); % instanciar o vetor yG
    % varrer as entradas de xG e para cada valor calcular o
    % valor que o spline interpolador toma
    for j = 1:size(xG,2)
  % calcular o número do troço a que a entrada atual de xG pertence.
       i = 1;
       while i<size(x,2)-1
          if(xG(j)<=x(i+1))
             break
          end
          i = i+1;
       end

       h = (x(i+1)-x(i));
       % Calcular o valor que a segunda derivada do spline toma
       %(cf. relatório)
       DDyG(j) = (M(i))*((x(i+1)-xG(j))/(h)) ...
                        + (M(i+1))*((xG(j)-x(i))/(h));
    end

end
```

Published with MATLAB® R2017b

```
function errorAnalysis(s,k,a,b)
% Descrição
% A função errorAnalysis tem como objetivo fazer o estudo do erro
 máximo
% cometido na interpolação por um spline em função do espaçamento
 entre
% troços (uniforme)
% Recebe como argumentos:
%   - s: o tipo de spline interpolador
%       - s=0: spline natural
%       - s=1: spline completo
%       - s=2: spline not a knot
%   - k: a função a usar na interpolação
%   - a,b: os extremos do intervalo onde a função fk(x) é interpolada

% Constantes
    % limitamos o número de troços por uma questão de memória e tempo
 de
    % execução, ao invés de limitar h
    n0 = 3; % número de troços inicial
    nMax = 1000; % número aproximado de troços máximo

% Calcular erro
% Sabemos que o erro máximo da interpolação varia com uma potência do
% espaçamento entre nós (cf. relatório). Assim, por forma a determinar
 essa
% potência iremos traçar um gráfico da dependência de log(E) com
 log(h)
% pelo que a pontência de h será o declive desse gráfico.
% Por forma a que os pontos que calculamos sucessivamente para o
 gráfico de
% log(E) em função log(h) tenham um espaçameto uniforme a variação de
 n
% (número de troços) será exponencial com um fator de 2

    % Instanciar os vetores h e E respetivamente os valores do
 espaçamento
    % entre troços em cada iteração
    h = zeros(ceil(log2(nMax/n0)),1);
    E = zeros(ceil(log2(nMax/n0)),3);

    % Calcular valores iniciais de h e E
    h(1) = (b-a)/n0;
    E(1,:) = splineUniform(s,k,a,b,n0,[1 1 1],0,0);

    % Varrer os valores de h e calcular o erro para cada um
    for i = 2:ceil(log2(nMax/n0))
        h(i) = 1/(n0*2^(i-1));
        E(i,:) = splineUniform(s,k,a,b,n0*2^(i-1),[1 1 1],0,0);
    end

% Traçar gráfico e estimar declive
```

```
    figure;
    loglog(h,E(:,1)); % traçar o gráfico numa escala logarítmica
    ylabel('log(|Erro_{max}|)  de f(x)');
    xlabel('log(h)');
    %detrminar o declive da reta
    slope = polyfit(log(h),log(E(:,1)),1);
    fprintf("Declive de log(E) vs log(h) para f(x) é %f com h a variar
 entre %f e %f\n", slope(1), (b-a)/3,  h(end));

    figure;
    loglog(h,E(:,2)); % traçar o gráfico numa escala logarítmica
    ylabel('log(|Erro_{max}|) de f´(x)');
    xlabel('log(h)');
    %detrminar o declive da reta
    slope = polyfit(log(h),log(E(:,2)),1);
    fprintf("Declive de log(E) vs log(h) para f'(x) é %f com h a
 variar entre %f e %f\n", slope(1), (b-a)/3,  h(end));

    figure;
    loglog(h,E(:,3)); % traçar o gráfico numa escala logarítmica
    ylabel('log(|Erro_{max}|)  de f´´(x)');
    xlabel('log(h)');
    %detrminar o declive da reta
    slope = polyfit(log(h),log(E(:,3)),1);
    fprintf("Declive de log(E) vs log(h) para f´´(x) é %f com h a
 variar entre %f e %f\n", slope(1), (b-a)/3,  h(end));
end
```

Published with MATLAB® R2017b

```
function M = Thomas(a,b,c,B)
% Função que recebe as diagonais principais, superior e inferior
% de uma matriz A tridiagonal e uma matriz coluna B e resolve o
% sistema de equações lineares AM=B em que A é diagonalmente
% dominante
% As referências para o método são indicadas no relatório,
% ainda que este resultado advenha do caso particular de eliminação
% de Gauss para matrizes tridiagonais diagonalmente dominantes.

    dim = length(B); % Obter a dimensão de A dimxdim
    beta = b(1) ;
    M(1) = B(1)/beta ;

    for j = 2:dim
        g(j) = c(j-1)/beta ;
        beta = b(j)-a(j)*g(j) ;
        M(j) = (B(j)-a(j)*M(j-1))/beta ;
    end

    % Substituindo de volta
    for j = 1:(dim-1)
        k = dim-j ;
        M(k) = M(k)-g(k+1)*M(k+1) ;
    end
end
```

Published with MATLAB® R2017b

```
function beam(filename,filenameSave)
% Descrição
% A função beamUniform interpola um conjunto de pontos (x,y) obtidos
% experimentalmente de um modo de vibração viga encastrada em ambas as
% extremidades, em que os pontos têm um espaçamento uniforme.
% Para a resolução deste problema usaremos um spline completo (cf.
% relatório) com derivadas nulas nas extremidades.
% A função recebe como input o nome do ficheiro no qual estão gurdados
% os valores de (x,y) da viga organizados em duas colunas e o nome do
% ficheiro em que pode ser guardado o spline. Se este nome for
% inválido não é guardado

% Constantes
    nG = 500; % número de pontos usados para o traçar o gráfico da
 viga

% Importar pontos
    fid = fopen(filename,'rt'); % abrir o fichero de texto dos dados
    % importar os valores e organiza-los numa matriz
    V = cell2mat(textscan(fid, '%f%f', 'MultipleDelimsAsOne',true ...
        ,'Delimiter',' ', 'HeaderLines',0));
    fclose(fid); % fechar o ficheiro

% Obter o vetor M
    % Sabemos que a derivada nos nós extremo de uma viga encastrada em
    % ambas as extremidades é nula nesses pontos pelo que D é nulo
    D = [0;0];
    % Calcular o vetor das segundas derivadas do spline interpolador
 em
    % cada um dos nós
    % Usando a função splineClampedAdaptive é possível interpolar
    % corretamente o modo de vibração da viga para nós com espaçamento
 não
    % uniforme
    M = splineClampedAdaptive(V(:,1)',V(:,2)',D);


% Criar gráfico
    xG = [0:1/nG:1]; % instanciar o vetor com os x dos pontos do
 gráfico
    % calcular os valores do spline interpolador nos pontos x onde
    % pretendemos traçar o gráfico a partir da função splineFunction
    yG = splineFunctionAdaptive(xG,V(:,1)',V(:,2)',M);
    figure;
    plot(xG,yG); % plot do spline interpolador
    hold on;
    scatter(V(:,1),V(:,2)); % plot dos valores experimentais
    title('Deslocamento vertical da viga')
    xlabel('x [m]');
    ylabel('w(x)');
    hold off;
```

```
% Criar gráfico da primeira derivada
    DyG = splineFunctionD(xG,V(:,1)',V(:,2)',M);
    figure;
    plot(xG,DyG); % plot do spline interpolador
    hold on;
    title('Rotação da viga')
    xlabel('x [m]');
    hold off;

% Criar gráfico da segunda derivada
    DDyG = splineFunctionDD(xG,V(:,1)',M');
    figure;
    plot(xG,DDyG); % plot do spline interpolador
    hold on;
    title('Curvatura da viga');
    xlabel('x [m]');
    hold off;

    %Só guarda o spline num ficheiro se o nome for válido
    if (isa(filenameSave, 'string') || isa(filenameSave, 'char') ...
                                        && ~strcmp(filenameSave,""))
        saveSpline(V(:,1)',V(:,2)',M,filenameSave);
    end
end
```

Published with MATLAB® R2017b

```
function E = splineAdaptive(k,a,b,epsl,PlotGraph)
% Descrição
% A função splineAdative interpola uma função dada pela função
% splineFunction através de um spline completo, adaptativamente. Dito
% por outras palavras, esta função interpola uma função num conjunto
% ótimo de nós e menor possível de forma a que o erro seja inferior
% a epsl.
% A função recebe como input:
%   -k: a função a interpolar
%   -a,b: os extremos do intervalo a interpolar
%   -epslon: o eero máximo permitido na interpolação
%   -PlotGraph:
%       -0: Não fazer o plot da função interpolada, do spline
% interpolador
%           nem dos nós usados para a interpolação
%       -1: Apesentar os resultados em um gráfico
% Esta função retorna o valor absoluto do maior erro cometido na
% interpolação.

% Funções Usadas e intervalos normalmente usados
% k -> fk(x)
% f1(x) = log(1+x)      em [0,1]
% f2(x) = 1/(1+25*x^2)  em [-1,1]
% f3(x) = sin(x^2)      em [0, 3*pi/2]
% f4(x) = tanh(x)       em [-20,20]

% Constantes
    % número de pontos usados para o tracar o gráfico do spline
    nG = 1000;

% Calcular nós
    % A função knotsAdaptiveSpline seleciona um conjunto de pontos
    % ótimo para interpolar a função no intervalo a,b e com erro
    % inferior a epsl
    [x,y,D] = knotsAdaptiveSpline(k,a,b,epsl);


% Obter o vetor M
% O vetor M, o vetor com o valor das segundas derivadas do spline
%interpolador em cada um dos nós

    % Resolver o sistema para cada tipo de spline

    % obter as derivadas da função interpolada nos nós extremos
    M = splineClampedAdaptive(x,y,D);

% Criar gráfico
    % instanciar o vetor com os x dos pontos do gráfico
    xG = [a:(b-a)/nG:b];
    % calcular os valores do spline interpolador nos pontos x onde
    % pretendemos traçar o gráfico a partir da função splineFunction
    yGs = splineFunctionAdaptive(xG,x,y,M);
```

```
    % calcular os valores da função interpolada nos pontos x do
 gráfico
    yGf = funChoice(xG,k,0);

    if(PlotGraph)
        figure;
        plot(xG,yGs); % plot do spline interpolador
        hold on;
        scatter(x,y); % plot da função interpolada
        plot(xG,yGf); % plot dos nós usados para a interpolação
        hold off;
    end

% Análise de erro
% Análise do erro cometido na interpolação e cálculo de E
    % vetor do erro de cada ponto calculado para o gráfico
    yGe = yGf-yGs;
    % valor absoluto do maior dos erros cometidos na interpolação
    E = max(abs(yGe));
    if(PlotGraph)
        figure;
        plot(xG,yGe); % plot da função de erro na interpolação
        hold off;
        % Print do valor do erro máximo na command window
        fprintf("Erro máximo na interpolação é: %f\n", E);
    end


end
```

Published with MATLAB® R2017b

```
function M = splineClampedAdaptive(x,y,D)
% Descrição
% A função splineClamped resolve o sistema de eqauções lineares
% necessário para determinar o valor da sengunda derivada de cada
% um dos nós do spline completo interpolador, com espaçamento
% não uniforme
% A função recebe como input:
%   -x: o vetor de nós a usar na interpolação
%   -D: vetor (2x1) que corresponde às derivadas da função
% interpolada nos nós extremos
%   -y: o vetor dos valores da função interpolada em cada nó
% Retorna o vetor M das segundas derivadas em cada um dos nós

% Criar matriz spline completo
% Podemos reduzir o problema do spline completo à resolução de
% um sistema de equações lineares representado por uma matriz
% tridiagonal de dimensões n+1 x n+1 (cf. relatório)

    n = size(x,2);
    A = zeros(n);

     %Matriz A - não é necessária (boa indicação para a
     % criação das diagonais)
     %A(1,1:2) = [2 1]; %condições de fecho
     %A(n,n-1:n) = [1 2];
     %for i = 2:n-1
        %A(i,i-1:i+1) = [x(i)-x(i-1) 2*(x(i+1)-x(i-1)) x(i+1)-x(i)];

    %end

    B = zeros(n,1);
    %condições de fecho
    B(1) = (6/(x(2)-x(1)))*((y(2)-y(1))/(x(2)-x(1))-D(1));
    B(n) = (6/(x(n)-x(n-1)))*(D(2)-(y(n)-y(n-1))/(x(n)-x(n-1)));

    for i = 2:n-1
        B(i) = 6*((y(i+1)-y(i))/(x(i+1)-x(i))-...
                                (y(i)-y(i-1))/(x(i)-x(i-1)));
    end

    % vetrores representativos das diagonais inferior, superior e
    % principal, respetivamente
    a = zeros(n,1);
    c = zeros(n,1);
    b = zeros(n,1);

    % Contruir diagonais (cf. relatório)
    for i = 2:n
       a(i) =  x(i)-x(i-1);
    end
```

```
    for i = 1:n-1
       c(i) =  x(i+1)-x(i);
    end

    for i = 2:n-1
        b(i) = 2*(x(i+1)-x(i-1));
    end

    % Condições fronteira
    c(1) = 1;
    a(n) = 1;
    b(1) = 2;
    b(n) = 2;

    % resolver o sistema de equações lineares
    %M = A\B;
    M = Thomas(a,b,c,B);


end
```

Published with MATLAB® R2017b

```
function [x,y,D] = knotsAdaptiveSpline(k,a,b, epsl)
% Descrição
% Esta função está responsável pela seleção dos nós ótimos para a
% interpolação de uma função f para um algoritmo adaptativo. Na
% verdade, como indicado no relatório usaremos uma aproximação para
% o majorante do erro. Verificamos que para apenas 2 troços inicais
% a aproximação é válida o suficiente para a aplicação da aproximação.
% É claro que para funções mais exóticas é necessário começar com um
% número de troços mais adequado

    % Criar partição inicial com 2 troços e pontso interiores
    % auxiliares
    x_ = [a:(b-a)/8:b];
    y_ = funChoice(x_,k,0);

    % Criar partição inicial com 2 troços sem os pontso auxilires
    x = [a:(b-a)/2:b];
    y = [y_(1) y_(5) y_(9)];

    % soma ponderada dos pontos usada para calcular a estimativa da
    %quarta derivada em ambos os troços
    d4Est1 = estimateD4(1,y_);
    d4Est2 = estimateD4(5,y_);

    % estimativa do majorante do erro
    error1 = abs((3125/385)*d4Est1);
    error2 = abs((3125/385)*d4Est2);

    % Verificar se cada um dos troços induz um erro superior a epsl
    % Nesse caso divide o intervalo em dois
    if (error2 > epsl)
        [x,y,x_,y_] = divideInterval(x,2,y,epsl,k,x_,y_);
    end
    if(error1 > epsl)
        [x,y,x_,y_] = divideInterval(x,1,y,epsl,k,x_,y_);
    end

    % calcular a derivada da função nos nós extremos a partir
    % de diferenças finitas
    D = derivativesExterior(x_,y_);

end

function [x,y,x_,y_] = divideInterval(x,idx,y,epsl,k,x_,y_)
% Função recursiva que divide um intervalo dado em dois e verifica se
% nesses dois novos intervalos o erro cometido é inferior a epsl. Se
% não for, a função invoca-se a ela mesma até que o erro seja
% inferior a epsl
% Recebe como argumentos:
% - os vetores de nós x e valores respetivos da função
% - os vetores de nós auxiliares e respetivos valores da função
% - o índice no vetor x do primeiro nó no extremo que se pretende
```

```
% dividir
% - k a função usada
% - epsl o limite maximo do erro


    % Calcular os pontos auxiliares em cada intervalo
    [x,y,x_,y_] = updateSupportVectors(x,y,x_,y_,idx,k);

    % Calculate error for new interval
    h = zeros(1,size(x,2)-1);
    for i = 2:size(x,2)
       h(i-1) = x(i)-x(i-1);
    end

    % Soma ponderada dos pontos usada para calcular a estimativa
    % da quarta derivada em ambos os troços
    d4Est1 = estimateD4(find(abs(x_-x(idx))<eps),y_);
    d4Est2 = estimateD4(find(abs(x_-x(idx+1))<eps),y_);

    % Aproximar o erro
    error1 = (3125/385)*d4Est1;
    error2 = (3125/385)*d4Est2;

    if (abs(error2) > epsl) % verifica se o erro é já inferior a epsl
        [x,y,x_,y_] = divideInterval(x,idx+1,y,epsl,k,x_,y_);
    end
    if (abs(error1) > epsl)
        [x,y,x_,y_] = divideInterval(x,idx,y,epsl,k,x_,y_);
    end
end

function [x,y,x_,y_] = updateSupportVectors(x,y,x_,y_,idx,k)
% Função que é chamada quando um intervalo é dividido por forma a
% atualizar os valores dos vetores de nós e de nós auxiliares na
% divisão do troço cujo índice do primeiro nó é idx
% Recebe como argumentos:
% - os vetores de nós x e valores respetivos da função
% - os vetores de nós auxiliares e respetivos valores da função
% - o índice no vetor x do primeiro nó no extremo que se pretende
% dividir
% - k a função usada

    % Introduzir um nó no vetor x
    x = [x(1:idx) (x(idx)+x(idx+1))/2 x(idx+1:end)];
    % Introduzir o valor respetivo da função no novo nó. è de notar
    % que não é necessário calcular esse valor novamente uma vez que
    % é um dos nós auxiliares
    y = [y(1:idx) y_(abs(x_-(x(idx+2)+x(idx))/2)< eps) y(idx+1:end)];

    % Determinar o índice de x_ do primeiro nó do troço considerado
    idx_ = find(abs(x_-x(idx))<eps);

    % Determinar os novos nós auxiliares
    for i = 1:4
```

```
        x_ = [x_(1:idx_) (x_(idx_)+x_(idx_+1))/2 x_(idx_+1:end)];
        y_ = [y_(1:idx_) funChoice((x_(idx_)+x_(idx_+2))/2,k,0)...
                                                    y_(idx_+1:end)];
        idx_ = idx_ +2;
    end

end

function d4Est = estimateD4(idx_,y_)
%Função que dado o índice do vetor y_ do primeiro nó do troço
% considerado retorna a soma ponderada que é usada para aproximar a
% quarta derivada no troço
    d4Est = y_(idx_)-4*y_(idx_+1)+6*y_(idx_+2)-4*y_(idx_+3)+...
                                                        y_(idx_+4);
end

function D = derivativesExterior(x,y)
% Função que utiliza diferencas finitas para calcular um valor
% aproximado para a primeira derivada nos nós extremos a partir
% de 5 pontos
    D = zeros(1,2);
    D(1) = ((-50*y(1)+96*y(2)-72*y(3)+32*y(4)-6*y(5))/...
                                                    (24*(x(2)-x(1))));
    D(2) = ((50*y(end)-96*y(end-1)+72*y(end-2)-...
                    32*y(end-3)+6*y(end-4))/(24*(x(end)-x(end-1))));
end
```

Published with MATLAB® R2017b

```
function plotSplineFromTxt(filename)
% Descrição
% A função saveSpline serve para importar e desenhar um spline
% que foi guardado num ficheiro de texto através da função saveSpline.
% A função recebe como input:
%   -sfilename: o nome do ficheiro onde estão guardados os dados

% Importar pontos
    fid = fopen(filename,'rt'); % abrir o fichero de texto dos dados
    if(fid<=3) % falha na abertura do ficheiro
        fprintf("Erro na abertura do ficheiro %s\n", filename);
    end
    % importar os valores e organiza-los numa matriz
    D = cell2mat(textscan(fid, '%f%f
%f', 'MultipleDelimsAsOne',true ...
        ,'Delimiter',' ', 'HeaderLines',0));
    fclose(fid); % fechar o ficheiro

    if(max(size(D))==0) % Se não for possível extrair valores
        fprintf("Erro na abertura do ficheiro %s\n", filename);
    end

    nG = 1000;
% Criar gráfico

    % instanciar o vetor com os x dos pontos do gráfico
    xG = [D(1,1):(D(end,1)-D(1,1))/nG:D(end,1)];
    % calcular os valores do spline interpolador nos pontos x onde
    % pretendemos traçar o gráfico a partir da função
    % splineFunctionAdaptive uma vez que o espaçamento pode ser
    % variável
    yGs = splineFunctionAdaptive(xG,D(:,1)',D(:,2)',D(:,3)');
    % calcular os valores da função interpolada para os pontos x
    % do gráfico

    figure;
    plot(xG,yGs); % plot do spline interpolador
    t = xlabel('x');
    t.Color = 'blue';
end
```

Published with MATLAB® R2017b

```
function saveSpline(x,y,M, sfilename)
% Descrição
% A função saveSpline serve para guardar os dados, se o utilizador
% assim o pretender
% A função recebe como input:
%   -x: vetor de nós de interpolação
%   -y: vetor do valor da função em cada um dos nós
%   -M: o vetor dos valores da segunda derivada do spline interpolador
% em cada um dos nós
%   -sfilename: o nome do ficheiro onde se irão guardar os valores
% É de notar que o spline fica apenas completamente definido sabendo
% estes 3 vetores (x,y,M)

fid = fopen(sfilename, 'wt'); %abrir o ficheiro
for i = 1:size(x,2)
 % preencher duas colunas do ficheiro txt com os valores de x e de M
    if(i~=1)
       fprintf(fid, '\n');
    end
    % escrever no ficheiro os vetores em coluna
    fprintf(fid, '%f %f %f' ,x(i),y(i),M(i));
end
fclose(fid); % fechar o ficheiro

end
```

Published with MATLAB® R2017b