Análise Numérica
Instituto Superior Técnico
Outubro 2022

Projeto Computacional

Clara Pereira (99405)
Marta Sereno (99432)
Samuel Pearson (99441)

Conteúdo

Exercício 1 **2**
1. a) 2
1. b) 2
1. c) 2

Exercício 2 **4**
2. a) 4
2. b) 5
2. c) 6
2. d) 7

Exercício 3 **8**
3. a) 8
3. b) 9

Exercício 1

1. a)

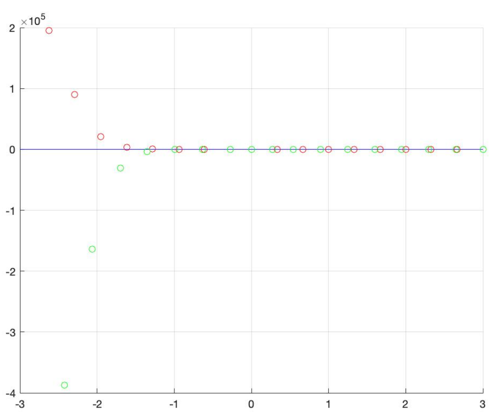
Para a resolução desta alínea, foram criadas três funções **divdif**,**Hermite**,**graficoh**. A função **divdif** calcula as diferenças divididas de uma dada função num dado vetor de x, devolvendo uma matriz. A função **Hermite** recebe também uma função e um vetor e calcula o polinómio de Hermite através da fórmula de Newton. A função **graficoh** recebe dois naturais n e M, a função f e a sua derivada df, e devolve um gráfico onde estão representadas f, df, e os respetivos polinómios de Hermite, h e dh, determinados com as funções anteriores. Na tentativa de escrever um programa genérico mas que nos facilitará mais tarde na representação da função enunciada, definimos que qualquer função f introduzida será sujeita a uma somatório, $symsum$, que por default toma $M = 1$. Foi necessário definir os vários valores que $x_k = \frac{3k}{n}$ devem tomar, com $k = 0, \ldots, n$, tendo sido definido o vetor $listax1$, que será utilizada como entrada de **Hermite**. Por fim, recorrendo a $fplot$, são representadas as quatro funções, f e df representadas a azul e h e dh representadas por pontos vermelhos e verdes, respetivamente. A determinação deste gráfico encontra-se no ficheiro **ex1a**, onde estão definidas as funções que se devem introduzir para valores de n e M à escolha.

1. b)

Analogamente à alínea anterior, foi criada a função **b1** (cujo script tem o mesmo nome), que recebe as mesmas entradas que a função da alínea anterior, calculando o Spline Cúbico para f e apresentando o gráfico do spline em $[0, 3]$ (colorido com várias cores, uma cor por cada intervalo $[x_k, x_{k+1}]$) e de f a vermelho, para comparação.
A função faz também o display do erro absoluto máximo obtido.

1. c)

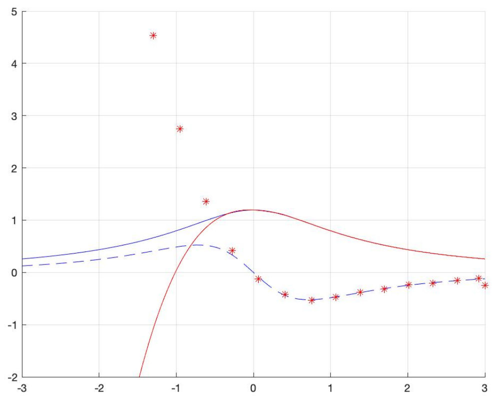
Apresentam-se os resultados da alínea a) para $(n, M) = (10, 1)$:



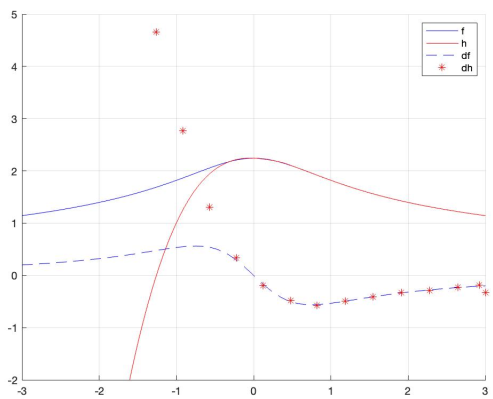
Note-se que correndo o programa para o intervalo $[0, 3]$ vê-se melhor ambas as funções e verifica-se que a aproximação é melhor.

O programa implementado em **graficoh** não apresenta, em tempo útil, o gráfico para n elevado, logo, optámos por usar o programa **hermitegraf** definido através da função $pchip$ do matlab, que determina o polinómio de Hermite. Então, para $(n, M) = (10, 10)$ e $(10, 1000)$ apresentamos os seguintes resultados da alínea a). A determinação deste gráfico encontra-se no ficheiro **ex1c**, onde estão definidas as funções que se devem introduzir para os valores de n e M.

$$(n, M) = (10, 10):$$



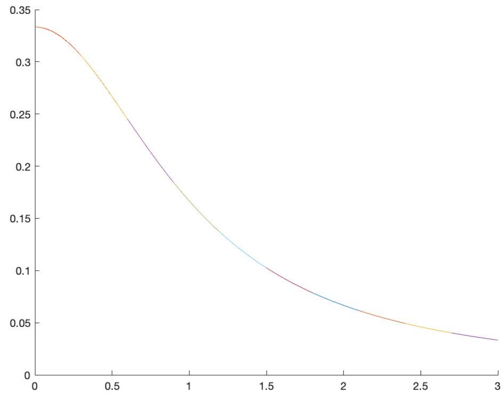
$$(n, M) = (10, 1000):$$

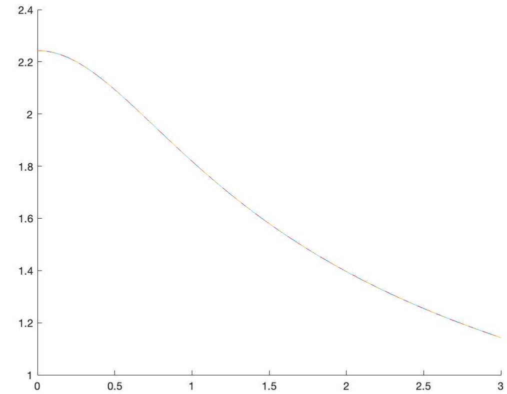


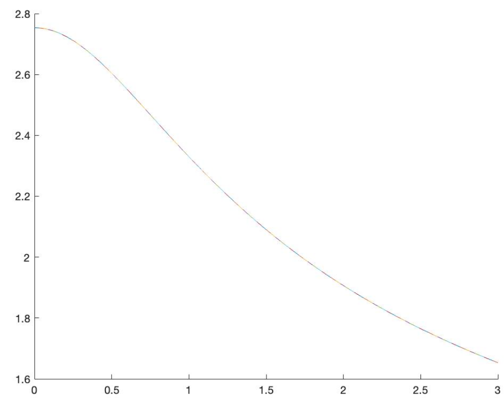
Na função **graficoh** definida para a alínea a), foram ainda calculados os erros absolutos dos polinómios de Hermite em comparação com as funções f e df. Não foi possivel calcular o erro teórico associado devido à sua complexidade computacional.
Verificou-se que com o aumento de n o erro diminui mas M não tem uma influência significativa no erro. Para $(10, 10)$ o erro foi 0.0084 e para $(100, 10)$ foi 3.5636e-05. No entanto, para $(10, 100)$ o erro foi semelhante ao de $(10, 10)$: 0.0078.

Como podemos observar pelos gráficos acima, no intervalo $[-3, 0)$ os polinómios diferem das funções enunciadas, devido ao facto das funções $Matlab$ definidas apenas interpolarem a função no intervalo $[0, 3]$.

Apresentam-se os resultados da alínea b) para alguns valores de n e M - $(n, M) = (10, 1), (100, 1000), (10, 10^5)$, (por ordem da esquerda para a direita e de cima para baixo) usando para f e df a função do enunciado e a sua derivada.







Os erros obtidos nos primeiros 2 casos foram, respetivamente, 1.8801e-04 e 2.5716e-08.
Para $(n, M) = (10, 100)$ e $(10, 1000)$ os erros foram, em ambos os casos, 2.8659e-04.
Isto leva-nos a concluir que o valor de n terá uma maior influência na qualidade da aproximação, sendo que um aumento de n num fator de 10 levou a uma diminuição do erro num fator de 10^4. Por outro lado, a variação de M não teve uma influência significativa no erro obtido.
Infelizmente, para $M = 10^6$, o programa dava erro devido ao elevado tamanho da expressão simbólica de f e df.
É visualmente aparente que, em todos os casos, a aproximação obtida será muito próxima da função real, uma vez que não se consegue discernir o gráfico de f que está "por trás"do gráfico do spline.

Exercício 2

2. a)

Para a resolução desta alínea, foi criada a função **Ex2a**, que recebe como argumentos N e E, e calcula o produto de convolução discreta $\tilde{\mathbf{f}} * \mu^{[0]}$, associado ao filtro $\mu_E^{[0]}$, em que $\tilde{\mathbf{f}}$ corresponde ao vetor de dimensão N apresentado no enunciado. A função começa então por calcular o vetor $\tilde{\mathbf{f}}$, e, para cada k, calcula a componente k do vetor v resultante da convolução, que é dado por:

$$[v]_k = \frac{1}{2E+1} \sum_{j=k-E}^{k+E} \tilde{f}_j$$

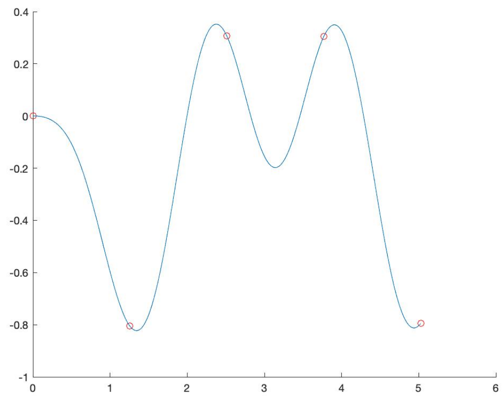
É preciso então, para cada k, calcular os índices j intervenientes no somatório acima. Por fim, a função devolve o vetor v.
Testou-se então as função para alguns valores de N e E, que se apresentam na tabela abaixo:

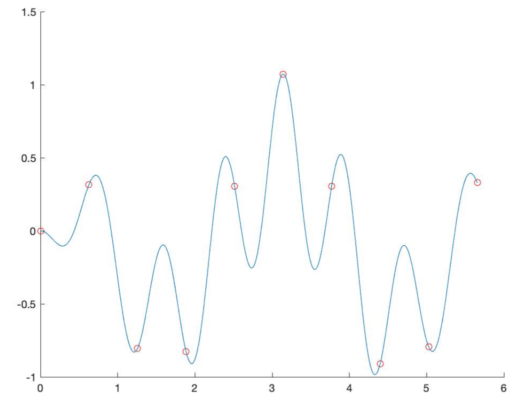
N	E	v
10	4	[-0.2309, -0.1457, -0.0108, -0.0235, -0.1483, -0.1118, -0.1470, -0.0223, -0.0200, -0.1458]
10	2	[-0.1909, -0.1972, -0.2017, 0.0128, 0.0104, -0.0103, -0.0040, 0.0005, -0.2140, -0.2116]
20	5	[0.0152, -0.1582, -0.0538, -0.0523, -0.1601, 0.0165, -0.0551, 0.0471, 0.0448, -0.0670, 0.1045, -0.0696, 0.0319, 0.0296, -0.0773, 0.0006, -0.1714, -0.0710, -0.0695, -0.1692]
20	2	[-0.2089, -0.2104, -0.1974, 0.0265, 0.0251, 0.0325, 0.0296, 0.0193, 0.0098, 0.0175, 0.0077, 0.0048, -0.0056, -0.0039, -0.0053, -0.0082, -0.0008, 0.0068, -0.2093, -0.2142]
50	10	[-0.1146, -0.0737, 0.0022, 0.0524, 0.0230, -0.0636, -0.0966, -0.0759, 0.0091, 0.0738, 0.0630, 0.0335, -0.0392, -0.0421, 0.0178, 0.0870, 0.0960, 0.0347, -0.0447, -0.0734, -0.0165, 0.0692, 0.0934, 0.0513, -0.0384, -0.0846, -0.0425, 0.0181, 0.0698, 0.0391, -0.0453, -0.1026, -0.0815, 0.0013, 0.0676, 0.0683, -0.0129, -0.0882, -0.0866, -0.0182, 0.0123, 0.0200, -0.0435, -0.1156, -0.1249, -0.0618, 0.0229, 0.0564, 0.0115, -0.0733]
50	5	[-0.1917, -0.1425, -0.0157, 0.0788, 0.0327, -0.0890, -0.0831, -0.0622, 0.0484, 0.1395, 0.0991, 0.0371, -0.0597, -0.0648, 0.0273, 0.1361, 0.1528, 0.0506, -0.0491, -0.1000, -0.0081, 0.1129, 0.1081, 0.0438, -0.0786, -0.1397, -0.0772, 0.0313, 0.1107, 0.0554, -0.0559, -0.1481, -0.1117, 0.0073, 0.0951, 0.0810, -0.0283, -0.1347, -0.1251, -0.0180, 0.1036, 0.1288, 0.0256, -0.0916, -0.1241, -0.1298, -0.0151, 0.0355, -0.0339, -0.1302]

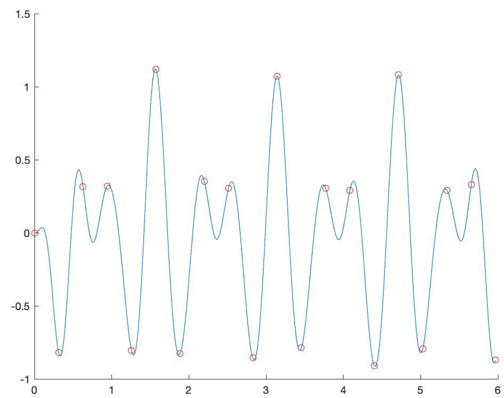
2. b)

Para a alínea b) do exercício 2, criou-se, em primeiro lugar, a função auxiliar **fourierd**, que calcula a Transformada de Fourier Discreta de um vetor de pontos y. Para calcular a função interpoladora trigonométrica do vetor $\tilde{\mathbf{f}}$, de dimensão N, recorremos à sua Transformada de Fourier. Desta forma, a função interpoladora será dada por $\phi(t) = \frac{1}{N}\mathcal{F}f \cdot u(t)$, em que $\mathcal{F}f$ corresponde à TFD do vetor $\tilde{\mathbf{f}}$, e $u(t) = 1, \exp i * t, ..., \exp it(N-1)$.
Criou-se então a função **Ex2b**, que recebe N, a dimensão de $\tilde{\mathbf{f}}$, e devolve a função que interpoladora esses pontos: começando por calcular o vetor $\tilde{\mathbf{f}}$, e aplicando-lhe a função **fourierd**. De seguinda calcula a lista de funções $u(t)$, e computa o polinómio final $\phi(t)$, pela fórmula apresentada acima. Uma vez que a função resultante é um somatório de exponenciais com N parcelas, o grupo achou relevante que a função devolvesse, para além da expressão do polinómio interpolador, também o seu gráfico entre os pontos interpolados, para ter uma perceção visual dos dados.

Em seguida apresentam-se os gráficos obtidos para $N = 5, 10, 20$.







2. c)

Para esta alínea, foi criada a função **fourierf**, que executa a Transformada Rápida de Fourier por um processo recursivo, descrito nos Apontamentos da Cadeira. É importante referir que este método apenas funciona se $N = 2^M$, para algum $M \in \mathbb{N}$. Para testar o tempo de execução, em segundos, de **fourierf** e compará-lo com o de **fourierd**, criou-se a função **Ex2c**, que recebe N, e devolve o tempo de execução destas duas funções para o vetor $\tilde{\mathbf{f}}$. Os resultados encontram-se na tabela seguinte.

N	$t(TFD)$	$t(FFT)$
2^3	0.0009	0.0017
2^6	0.0007	0.0018
2^{10}	0.0313	0.0090
2^{15}	16.3098	0.3370
2^{16}	83.5641	0.4738
2^{17}	366.9733	0.6635

Como resultado teórico, temos que o número de operações realizadas na Transformada Rápida é $O(Nlog_2(N))$, enquanto que na Transformada Discreta o número de operações é $O(N^2)$. Adicionalmente, o método TFD realiza mais multiplicações, que são mais complexas do que somas. Assim, os resultados corroboram a teoria, uma vez que para N elevado a função **fourierf** é executada em muito menos tempo do que **fourierd**, o que seria de esperar, pois a qauntidade e complexidade de operações envolvidas é muito menor.

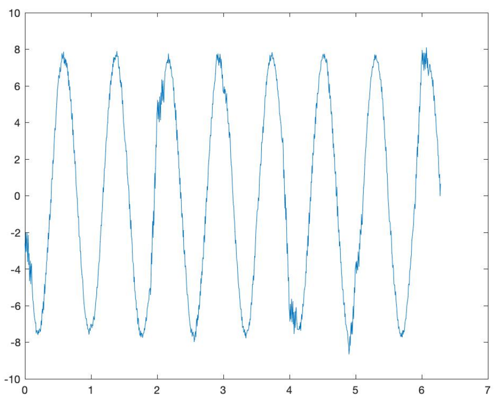
2. d)

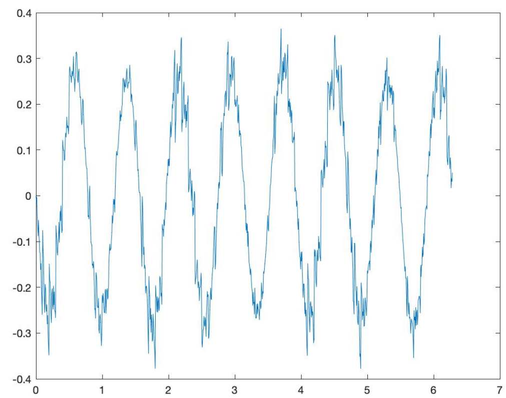
Implementou-se para esta alínea a função **D2**, com o intuito de obter uma expressão para a aproximação da função f' (e produzir um gráfico). A função recebe apenas o input "*eps*"de modo a definir os extremos de integração para cada y, sendo n fixo com valor 20. Tendo em conta que

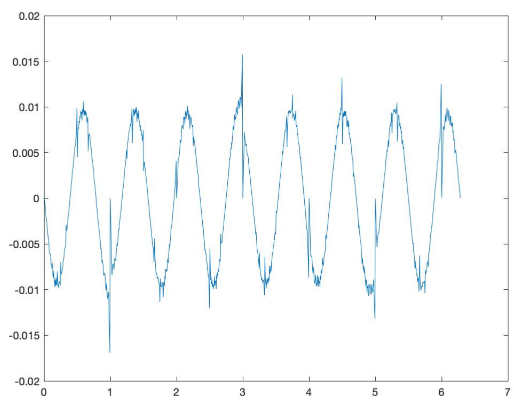
$$f'(y) \simeq \langle \mu_{\epsilon,y}^{[1]}, f' \rangle_{L^2(\mathbb{R})} = -\langle \mu_{\epsilon,y}^{[1]'}, f \rangle_{L^2(\mathbb{R})} = -\frac{1}{\epsilon^2}(\int_{y-\epsilon}^{y} f(t)dt - \int_{y}^{y+\epsilon} f(t)dt)$$

teremos então que aproximar os integrais da expressão, usando a Regra dos Trapézios Composta. O código encontra-se devidamente comentado no script *D2.m*.

Apresentam-se os gráficos obtidos para os inputs 0.1, 1 e 10 (da esquerda para a direita e de cima para baixo):







É de notar que quanto mais pequeno o input ϵ, maior é a escala no gráfico obtido. Isto é de esperar, uma vez que, olhando para o gráfico, o valor da derivada da função interpoladora de f oscila bastante, tomando valores por vezes elevados.
Assim sendo, para ϵ grande, a aproximação de f' ficará mais regular devido ao aumento do intervalo de integração e para ϵ menor será à partida mais próxima do valor real.
Todavia, há que ter em atenção que a divisão por ϵ^2 no cálculo da aproximação da derivada pode originar problemas, pois para valores muito pequenos de ϵ podemos obter valores astronómicos para f' em certos pontos.

Exercício 3

3. a)

O número mecanográfico escolhido foi 99432, com 4 dígitos diferentes: $n_1 = 9, n_2 = 4, n_3 = 3, n_4 = 2$. Assim sendo, foi necessário determinar uma fórmula para $f''(z)$ com base nos valores $f_k = f(z + (5 - n_k)h)$.

Escrevendo cada f_k em termos do polinómio de Taylor de f centrado em z temos:

- $f_1 = f(z-4h) = f(z) - f'(z)\cdot 4h + \frac{f''(z)\cdot(4h)^2}{2} - \frac{f'''(z)\cdot(4h)^3}{6} + \frac{f^{(4)}(\xi_1)\cdot(4h)^4}{24}$, $\xi_1 \in [z-4h, z]$
- $f_2 = f(z+h) = f(z) + f'(z)\cdot h + \frac{f''(z)\cdot(h)^2}{2} + \frac{f'''(z)\cdot(h)^3}{6} + \frac{f^{(4)}(\xi_2)\cdot(h)^4}{24}$, $\xi_2 \in [z, z+h]$
- $f_3 = f(z+2h) = f(z) + f'(z)\cdot 2h + \frac{f''(z)\cdot(2h)^2}{2} + \frac{f'''(z)\cdot(2h)^3}{6} + \frac{f^{(4)}(\xi_3)\cdot(2h)^4}{24}$, $\xi_3 \in [z, z+h]$
- $f_4 = f(z+3h) = f(z) + f'(z)\cdot 3h + \frac{f''(z)\cdot(3h)^2}{2} + \frac{f'''(z)\cdot(3h)^3}{6} + \frac{f^{(4)}(\xi_4)\cdot(3h)^4}{24}$, $\xi_4 \in [z, z+3h]$

Visto que queremos determinar $f''(z)$, podemos esperar encontrar coeficientes $c_1, ..., c_4$ tais que $\sum_{k=1}^{4} c_k \cdot f_k$ nos deixe com uma expressão sem termos de ordem 0, 1 ou 3, de modo a "isolar"(quase) os termos de ordem 2. (Note-se que para eliminarmos também os termos de ordem 4 teríamos um sistema linear homogéneo de 4 equações linearmente independentes e 4 incógnitas, cuja única solução seria $(0, 0, 0, 0)$).
Obtemos assim um sistema de equações lineares:

$$\begin{cases} c_1 + c_2 + c_3 + c_4 = 0 \\ -4c_1 + c_2 + 2c_3 + 3c_4 = 0 \\ -64c_1 + c_2 + 8c_3 + 27c_4 = 0 \end{cases}$$

É simples verificar que a menor solução inteira deste sistema é $(c_1, c_2, c_3, c_4) = (2, -7, 0, 5)$.
Temos então:

$$\sum_{k=1}^{4} c_k \cdot f_k = 2\cdot f(z-4h) - 7\cdot f(z+h) + 5\cdot f(z+3h) =$$

$$35h^2 \cdot f''(z) + \frac{h^4}{4!}(512\cdot f^4(\xi_1) - 7\cdot f^4(\xi_2) + 405\cdot f^4(\xi_4))$$

Obtém-se a expressão resultante:

$$f''(z) = \frac{2\cdot f(z-4h) - 7\cdot f(z+h) + 5\cdot f(z+3h)}{35h^2} - \frac{h^2}{840}(512\cdot f^4(\xi_1) - 7\cdot f^4(\xi_2) + 405\cdot f^4(\xi_4))$$

O termo com as derivadas de ordem 4 corresponde ao erro. O módulo do erro pode ser facilmente majorado por

$$\frac{h^2}{840}(|512\cdot f^4(\xi)| + |-7\cdot f^4(\xi)| + |405\cdot f^4(\xi)|) = \frac{924h^2}{840}|f^4(\xi)|$$

onde $|f^4(\xi)| = max_{\eta \in [z-4h, z+3h]} |f^4(\eta)|$.

Consideremos agora que os valores f_1, f_2, f_3, f_4 estavam afetados de erros e que dispunhamos apenas dos respetivos valores aproximados $\tilde{f}_0, ..., \tilde{f}_4$.
Teríamos então:

$$f''(z) = \frac{2\tilde{f}_1 - 7\tilde{f}_2 + 5\tilde{f}_4}{35h^2} - \frac{h^2}{840}(512 \cdot f^4(\xi_1) - 7 \cdot f^4(\xi_2) + 405 \cdot f^4(\xi_4)) + \frac{2\epsilon_1 - 7\epsilon_2 + 5\epsilon_4}{35h^2}$$

onde $\epsilon_i = f_i - \tilde{f}_i$. Assim:

$$|f''(z) - \frac{2\tilde{f}_1 - 7\tilde{f}_2 + 5\tilde{f}_4}{35h^2}| \leq \frac{924h^2}{840}|f^4(\xi)| + \frac{16\epsilon}{35h^2}$$

onde $\epsilon = max_{k=1,2,4}|\epsilon_k|$.
Claramente, $\frac{924h^2}{840}|f^4(\xi)| \to 0$ quando $h \to 0$.
No entanto, o mesmo não é necessariamente verdade para o termo $\frac{16\epsilon}{35h^2}$, sendo necessário que ϵ acompanhe o decrescimento de h^2, isto é, que $\epsilon = \mathcal{O}(h^2)$, para que a aproximação não perca a eficácia.

3. b)

Para aplicar a fórmula deduzida na alínea anterior, impementou-se a função **Ex3b** (com o mesmo nome do script em que se encontra), que recebe como argumentos h e b. Se o input b for igual a 1, a função f a considerar é a sugerida no enunciado. Caso contrário, será a função escolhida pelo grupo, $f(x) = \frac{sin(x^4)}{x}$.

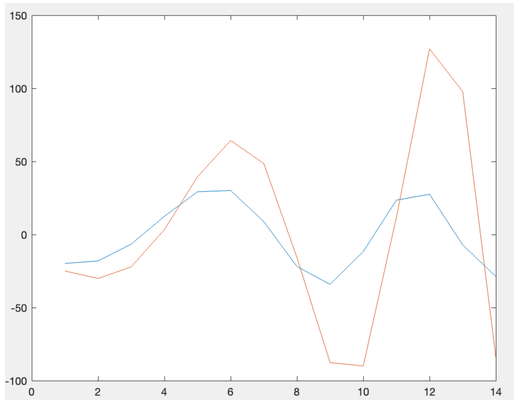
A função **Ex3b** calculará então os valores de $f''(z)$ (com $z = kh \in [0, 1 + \frac{3}{10}]$) pelo método da alínea a) (usando a função *aprox*) e pela própria função do Matlab dada pela respetiva expressão, ficando estes valores guardados nas listas *f2vals* e *f2real_vals*, respetivamente.

De seguida, cria-se a lista *erro_abs*, correspondendo ao módulo da diferença entre os valores de $f''(z)$, para cada z, obtidos com cada método.

Dado z, a função *aux* calcula $\frac{924h^2}{840}|f^4(\xi)|$ com ξ sendo um ponto maximizante de $|f^4(x)|$ em $[z - 4h, z + 3h]$, sendo o intervalo discretizado de forma suficientemente fina, e não contínuo.
Com o auxílio desta função, cria-se a lista *majorante_erro* que contém os majorantes dos erros obtidos no cálculo de $f''(z)$ para cada $z \in [0, 1 + \frac{3}{10}]$.

Testou-se em primeiro lugar a função $f(x) = cos(3x^2 + 5x)$ para diferentes valores de h, tendo-se obtido os seguintes resultados:

h=0.1 (os comandos executados foram *[a,b,c,d]=Ex3b(0.1,1);* seguido de *hold on; plot(a,'b'); plot(b,'r');*, de modo a obter um gráfico dos valores de *f2vals* a azul e *f2real_vals* a vermelho) :



Claramente que $h = 0.1$ não é suficientemente pequeno para obtermos uma boa estimativa de $f''(z)$.

Para os seguintes valores de h apresentamos numa tabela apenas os valores $max(erro\_abs)$ e $max(majorante\_erro)$, para ambas as funções:

h	b	$max(erro\_abs)$	$max(majorante\_erro)$
0.01	1	2.2293	2.2783
0.01	0	0.6896	0.7383
0.001	1	0.0224	0.0228
0.001	0	0.0070	0.0072
0.0001	1	2.2439e-04	2.2783e-04
0.0001	0	6.9753e-05	7.0903e-05

Após a observação destes dados, pode-se verificar que quando se divide h por 10, o erro absoluto diminui por um fator de aproximadamente 10^2, pelo que a convergência é de ordem $\mathcal{O}(h^2)$.
O máximo do erro absoluto (e do majorante) manteve-se aproximadamente 10 vezes menor para a função escolhida pelo grupo do que para a função sugerida.
Pode-se também aferir que o majorante de erro parece ser uma boa aproximação do erro real para estas duas funções.