Instituto Superior Técnico

Matemática Computacional

3º Quarter - 2021/2022

Licenciatura Em Matemática Aplicada e Computação

Segundo Projeto Computacional

Abril 2022

Clara Pereira 99405
Marta Sereno 99432
Samuel Pearson 99441

Conteúdo

Exercício 1

1. a)

O objetivo desta alínea é provar as igualdades $J_F(x)^T F(x) = \nabla g$
e $H_g(x) = J_F(x)^T J_F(x) + S(x)$.
Comecemos por provar que $J_F(x)^T F(x) = \nabla g(x)$.
A matriz Jacobiana de F e a sua transposta são dadas por:

$$J_F(x) = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1} & \cdots & \frac{\partial f_1(x)}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_M(x)}{\partial x_1} & \cdots & \frac{\partial f_M(x)}{\partial x_n} \end{bmatrix}, J_F(x)^T = \begin{bmatrix} \frac{\partial f_1(x)}{\partial x_1} & \cdots & \frac{\partial f_M(x)}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_1(x)}{\partial x_n} & \cdots & \frac{\partial f_M(x)}{\partial x_n} \end{bmatrix}$$

Calculando o gradiente de g, obtemos:

$$\nabla g(x) = \left(\frac{\partial}{\partial x1} \frac{1}{2} \sum_{j=1}^{M} f_j^2(x), \quad \cdots \quad , \frac{\partial}{\partial xn} \frac{1}{2} \sum_{j=1}^{M} f_j^2(x) \right) =$$

$$= \left(\sum_{j=1}^{M} f_j(x) \frac{\partial f_j(x)}{\partial x_1}, \quad \cdots \quad , \sum_{j=1}^{M} f_j(x) \frac{\partial f_j(x)}{\partial x_n} \right)$$

Assim,

$$J_F(x)^T F(x) = \left(\sum_{j=1}^{M} f_j(x) \frac{\partial f_j(x)}{\partial x_1}, \quad \cdots \quad , \sum_{j=1}^{M} f_j(x) \frac{\partial f_j(x)}{\partial x_n} \right) = \nabla g(x)$$

Provando, desta forma, a primeira parte da alínea.
Queremos agora provar que $H_g(x) = J_F(x)^T J_F(x) + S(x)$. Temos então:

$$H_{f_j}(x) = \begin{bmatrix} \frac{\partial^2 f_j(x)}{\partial x_1^2} & \cdots & \frac{\partial^2 f_j(x)}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial^2 f_j(x)}{\partial x_n \partial x_1} & \cdots & \frac{\partial^2 f_j(x)}{\partial x_n^2} \end{bmatrix}$$

Logo,

$$S(x) = \sum_{j=1}^{M} f_j H_{f_j} = \begin{bmatrix} \sum_{j=1}^{M} f_j(x) \frac{\partial^2 f_j(x)}{\partial x_1^2} & \cdots & \sum_{j=1}^{M} f_j(x) \frac{\partial^2 f_j(x)}{\partial x_1 \partial x_n} \\ \vdots & \ddots & \vdots \\ \sum_{j=1}^{M} f_j(x) \frac{\partial^2 f_j(x)}{\partial x_n \partial x_1} & \cdots & \sum_{j=1}^{M} f_j(x) \frac{\partial^2 f_j(x)}{\partial x_n^2} \end{bmatrix}$$

Temos ainda que

$$J_F^T(x)J_F(x) = \begin{bmatrix} \sum_{j=1}^M [\frac{\partial f_j(x)}{\partial x_1}]^2 & \cdots & \sum_{j=1}^M \frac{\partial f_j(x)}{\partial x_1}\ \frac{\partial f_j(x)}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \sum_{j=1}^M \frac{\partial f_j(x)}{\partial x_n}\ \frac{\partial f_j(x)}{\partial x_1} & \cdots & \sum_{j=1}^M [\frac{\partial f_j(x)}{\partial x_n}]^2 \end{bmatrix}$$

Finalmente,

$$H_g(x) = \begin{bmatrix} \sum_{j=1}^M [[\frac{\partial f_j(x)}{\partial x_1}]^2 + f_j(x)\frac{\partial^2 f_j(x)}{\partial x_1^2}] & \cdots & \sum_{j=1}^M [\frac{\partial f_j(x)}{\partial x_1}\ \frac{\partial f_j(x)}{\partial x_n} + f_j(x)\frac{\partial^2 f_j(x)}{\partial x_1 \partial x_n}] \\ \vdots & \ddots & \vdots \\ \sum_{j=1}^M [\frac{\partial f_j(x)}{\partial x_n}\ \frac{\partial f_j(x)}{\partial x_1} + f_j(x)\frac{\partial^2 f_j(x)}{\partial x_n \partial x_1}] & \cdots & \sum_{j=1}^M [[\frac{\partial f_j(x)}{\partial x_n}]^2 + f_j(x)\frac{\partial^2 f_j(x)}{\partial x_n^2}] \end{bmatrix}$$

$$H_g(x) = J_F(x)^T J_F(x) + S(x)$$

1. b)

Sabemos que, sendo F duas vezes diferenciável, para que $z \in \mathbb{R}$ seja um mínimo de $\frac{1}{2}\|\tilde{F}(x, x^{(k)})\|_2^2$, é necessário que $\nabla \frac{1}{2}\|\tilde{F}(x, x^{(k)})\|_2^2 = 0$ em z.
Sabemos também, pela Alínea a), que $\nabla \frac{1}{2}\|\tilde{F}(x, x^{(k)})\|_2^2 = J_{\tilde{F}}(x, x^{(k)})\tilde{F}(x, x^{(k)})$.
Note-se que o Passo 1 do método de Gauss-Newton, que pretende encontrar pontos estacionários, corresponde a resolver o seguinte sistema linear:

$$J_F(x^{(k)})^T J_F(x)p^{(k)} = -J_F(x^{(k)})^T F(x^{(k)}) \Leftrightarrow J_F(x^{(k)})^T[F(x^{(k)}) + J_F(x^{(k)})p^{(k)}] = 0$$

Sendo $\tilde{F}$ a linearização de F em torno de $x^{(k)}$, temos que

$$J_F(x^{(k)}) = J_{\tilde{F}}(x, x^{(k)}), \forall x \in \mathbb{R}^N$$

Considerando $\tilde{z} = x^{(k+1)} = x^{(k)} + p^{(k)}$, temos $p^{(k)} = \tilde{z} - x^{(k)}$ e portanto no método de Gauss-Newton estamos a obter aproximações para a solução do sistema linear

$$J_{\tilde{F}}(\tilde{z}, x^{(k)})^T \tilde{F}(\tilde{z}, x^{(k)}) = 0$$

o que corresponde precisamente a aproximar pontos estacionários de $\frac{1}{2}\|\tilde{F}(x, x^{(k)})\|_2^2$.

1. c)

Para esta alínea, implementámos o método de Gauss-Newton e de Newton, com as funções $c1\_GN$ e $c1\_N$, respetivamente.

Notou-se também que a convergência depende de o x_0 escolhido estar suficientemente próximo da solução, pelo que se achou pertinente que, para além dos parâmetros pedidos no enunciado, ambos os códigos recebessem adicionalmente o parâmetro x_0.

É importante salientar que os métodos não encontram mínimos, mas apenas pontos estacionários. Assim, depois de concluído o método, é ainda necessário testar se a solução z encontrada é, de facto, um mínimo. Para isto, calculamos a matriz Hessiana de g, $H_g(x)$, que queremos que seja Definida Positiva. Podemos utilizar um dos resultados obtidos na Alínea a), $H_g(x) = J_F(x)^T J_F(x) + S(x)$.

Uma vez que a função g é par em x_1, para qualquer minimizante (z_1, z_2) encontramos também o minimizante $(-z_1, z_2)$.

Foram encontradas apenas duas soluções, $(1.3788, 0.9942)$ e $(-1.3788, 0.9942)$, para as quais o mínimo da função g é igual a 0.0219.

Obtiveram-se também vários pontos estacionários. No entanto não correspondiam a mínimos. Note-se que, ao longo da reta $x_1 = 0$ todos os pontos são estacionários, mas não são extremos, uma vez que a função é constante ao longo dessa reta:
$g(0, x_2) = \frac{1}{2} \sum_{j=1}^{5} y_j^2(x)$.

Apresenta-se, em baixo, o gráfico da função aproximadora $y(t) = z_1^2 e^{tz_2}$, correspondente ao método de Newton (e Gauss-Newton).

Observação: a função aproximadora é igual para ambas as soluções e ambos os métodos, pelo que apresentamos apenas gráfico para uma delas, pelo método de Newton.

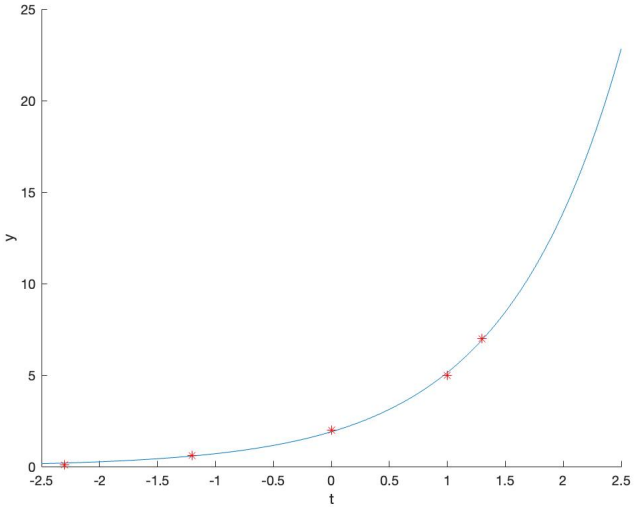

Figura 1: Função aproximadora e conjunto de pontos, pelo método de Newton

1. d)

Como pedido, realizaram-se ambos os métodos, desta vez num conjunto de seis pontos. Mais uma vez, encontraram-se duas soluções, com as primeiras componentes simétricas: $(2.0071, 1.4500)$ e $(-2.0071, 1.4500)$.

No entanto, ao contrário do que ocorreu na alínea c), o gráfico da função aproximadora dista bastante dos pontos (t_j, y_j). Isto deve-se ao facto de o sexto ponto não concordar com os restantes: a função aproximadora é monótona independentemente da solução (z_1, z_2), e temos $t_4 < t_6 < t_5$ mas $y_4 < y_5 <<< y_6$.

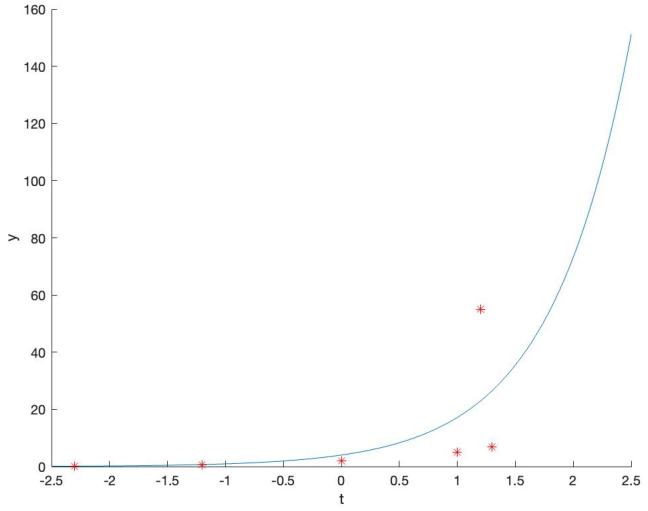

Figura 2: Função aproximadora e conjunto de pontos, pelo método de Newton

Exercício 2

2. a)

Nesta alínea, temos como objetivo definir uma matriz tridiagonal, com componentes

$$a_{jj} = 2\ , j = 1, \ldots, N, \quad a_{j,j-1} = -1\ , j = 2, \ldots, N, \quad a_{j+1,j} = -1\ , j = 1, \ldots, N-1$$

e determinar, para $N = 5, 10, 15, 20, 25, \ldots$ o número de condição de cada matriz. Para tal, foi criada a função *matrixN* que recebe a dimensão da matriz desejada, N, e os valores das componentes na diagonal principal, na diagonal diretamente em cima da principal, e na diagonal diretamente abaixo da principal, respetivamente *diag*, *ovr*, *und*. De seguida, foi criada a função *cond* que recebe uma matriz A e, através da determinação dos seus valores próprios e dos valores próprios da sua matriz inversa, A^{-1}, e determina o número de condição associado ao raio espetral de A, $cond(A) = r_\sigma(A) \cdot r_\sigma(A^{-1})$, onde $r_\sigma(A)$ é o raio espetral da matriz A.

N	Número de Condição
5	13.9282
10	48.3742
15	103.0869
20	178.0643
25	273.3061

Após a determinação dos valores, podemos observar que o condicionamento da matriz A depende de N - quanto maior N, maior é o seu número de condição.

2. b)

Em primeiro lugar, é-nos pedido que implementemos o método SOR no $Matlab$. Assim, criámos uma função SOR que recebe, tal como pedido, uma matriz A, um vetor b, um parâmetro ω, e o número máximo de iterações que o programa deverá realizar $nmax$.
Depois das verificações de aplicabilidade do $input$, começamos por dividir a matriz A em três novas matrizes, uma matriz diagonal D, uma matriz atómica triangular superior U e uma matriz atómica triangular inferior L. Definimos um valor inicial $x^{(0)} = (0, 0, \ldots, 0)$ com dimensão N. Recorrendo a um ciclo $while$, enquanto o número de iterações realizadas i não for superior ao número máximo de iterações a realizar, o programa calcula a próxima iteração, recorrendo à expressão do método. Assim que o ciclo acaba, o programa devolve a resposta final, a solução resultante da última iteração realizada.

Após a implementação do programa, é pedido que o utilizemos para aproximar as soluções $\tilde{z}$ do sistema $Ax = b$, onde A é uma matrix tridiagonal definida na alínea anterior e b é um vetor coluna com $b_j = 1$, $j = 1, \ldots, N$. Consideramos $N = 5, 10, 15, 20$, $nmax = 20$ e $\omega = 0.5, 0.75, 1.25, 1.5, 1.75$ e apresentamos, para cada N, os resultados para os vários valores de ω numa tabela. Calculamos também o erro absoluto e relativo para cada ω, tendo para isso definido a função $erros$.Considerando para o valor exato z a solução dada pelo comando $linsolve$, temos:

$$erro_{abs} = \|z - \tilde{z}\|$$

$$erro_{rel} = \frac{erro_{abs}}{\|z\|}$$

Para $N = 5, 10, 15, 20$:

	$\omega = 0.50$	$\omega = 0.75$	$\omega = 1.25$	$\omega = 1.50$	$\omega = 1.75$	linsolve
SOR	$\begin{bmatrix} 2.0768 \\ 3.2989 \\ 3.7257 \\ 3.3587 \\ 2.1460 \end{bmatrix}$	$\begin{bmatrix} 2.3955 \\ 3.8334 \\ 4.3228 \\ 3.8587 \\ 2.4248 \end{bmatrix}$	$\begin{bmatrix} 2.49999 \\ 3.9999 \\ 4.4999 \\ 4.0000 \\ 2.5000 \end{bmatrix}$	$\begin{bmatrix} 2.5000 \\ 4.0000 \\ 4.5000 \\ 4.0000 \\ 2.5000 \end{bmatrix}$	$\begin{bmatrix} 2.4937 \\ 3.9895 \\ 4.4910 \\ 4.0025 \\ 2.5020 \end{bmatrix}$	$\begin{bmatrix} 2.5000 \\ 4.0000 \\ 4.5000 \\ 4.0000 \\ 2.5000 \end{bmatrix}$
$erro_{abs}$	1.3442	0.3094	$1.2079 \cdot 10^{-4}$	$1.0014 \cdot 10^{-5}$	0.0155	-
$erro_{rel}$	0.1670	0.03845	$1.5011 \cdot 10^{-5}$	$1.2448 \cdot 10^{-6}$	0.0019	-

	$\omega = 0.50$	$\omega = 0.75$	$\omega = 1.25$	$\omega = 1.50$	$\omega = 1.75$	linsolve
SOR	$\begin{bmatrix} 2.3117 \\ 3.9023 \\ 4.9520 \\ 5.6025 \\ 5.9486 \\ 6.0314 \\ 5.8345 \\ 5.2830 \\ 4.2471 \\ 2.5533 \end{bmatrix}$	$\begin{bmatrix} 3.1577 \\ 5.5440 \\ 7.2755 \\ 8.4380 \\ 9.0803 \\ 9.2113 \\ 8.7998 \\ 7.7786 \\ 6.0508 \\ 3.4999 \end{bmatrix}$	$\begin{bmatrix} 4.6279 \\ 8.3353 \\ 11.1352 \\ 13.0312 \\ 14.0187 \\ 14.0867 \\ 13.2188 \\ 11.3959 \\ 8.5977 \\ 4.8049 \end{bmatrix}$	$\begin{bmatrix} 4.9759 \\ 8.9606 \\ 11.9531 \\ 13.9518 \\ 14.9553 \\ 14.9618 \\ 13.9701 \\ 11.9788 \\ 8.9871 \\ 4.9942 \end{bmatrix}$	$\begin{bmatrix} 5.0033 \\ 9.0491 \\ 12.0531 \\ 14.0551 \\ 15.0543 \\ 15.0502 \\ 14.0426 \\ 12.0323 \\ 9.0203 \\ 5.0087 \end{bmatrix}$	$\begin{bmatrix} 5.0000 \\ 9.0000 \\ 12.0000 \\ 14.0000 \\ 15.0000 \\ 15.0000 \\ 14.0000 \\ 12.0000 \\ 9.0000 \\ 5.0000 \end{bmatrix}$
$erro_{abs}$	21.3574	13.8909	2.2889	0.1080	0.1307	-
$erro_{rel}$	0.5830	0.3792	0.0625	0.0029	0.0036	-

	$\omega = 0.50$	$\omega = 0.75$	$\omega = 1.25$	$\omega = 1.50$	$\omega = 1.75$	linsolve
SOR	$\begin{bmatrix} 2.3131 \\ 3.9077 \\ 4.9695 \\ 5.6527 \\ 6.0776 \\ 6.3327 \\ 6.4784 \\ 6.5511 \\ 6.5652 \\ 6.5111 \\ 6.3507 \\ 6.0107 \\ 5.3773 \\ 4.2952 \\ 2.5738 \end{bmatrix}$	$\begin{bmatrix} 3.1848 \\ 5.6219 \\ 7.4536 \\ 8.8045 \\ 9.7793 \\ 10.4602 \\ 10.9053 \\ 11.1467 \\ 11.1877 \\ 11.0018 \\ 10.5312 \\ 9.6881 \\ 8.3572 \\ 6.4014 \\ 3.6686 \end{bmatrix}$	$\begin{bmatrix} 5.2694 \\ 9.7603 \\ 13.5357 \\ 16.6442 \\ 19.1193 \\ 20.9783 \\ 22.2229 \\ 22.8392 \\ 22.8000 \\ 22.0659 \\ 20.5881 \\ 18.3110 \\ 15.1745 \\ 11.1171 \\ 6.0780 \end{bmatrix}$	$\begin{bmatrix} 6.5998 \\ 12.3432 \\ 17.2432 \\ 21.3049 \\ 24.5265 \\ 26.8998 \\ 28.4119 \\ 29.0457 \\ 28.7812 \\ 27.5970 \\ 25.4708 \\ 22.3808 \\ 18.3063 \\ 13.2285 \\ 7.1310 \end{bmatrix}$	$\begin{bmatrix} 7.5696 \\ 14.1206 \\ 19.6520 \\ 24.1640 \\ 27.6581 \\ 30.1368 \\ 31.6041 \\ 32.0652 \\ 31.5267 \\ 29.9965 \\ 27.4837 \\ 23.9986 \\ 19.4953 \\ 13.9947 \\ 7.4965 \end{bmatrix}$	$\begin{bmatrix} 7.5000 \\ 14.0000 \\ 19.5000 \\ 24.0000 \\ 27.5000 \\ 30.0000 \\ 31.5000 \\ 32.0000 \\ 31.5000 \\ 30.0000 \\ 27.5000 \\ 24.0000 \\ 19.5000 \\ 14.0000 \\ 7.5000 \end{bmatrix}$
$erro_{abs}$	72.3297	59.0130	26.1164	8.6424	0.3595	-
$erro_{rel}$	0.7738	0.6313	0.2794	0.0925	0.0038	-

	$\omega = 0.25$	$\omega = 0.50$	$\omega = 0.75$	$\omega = 1.00$	$\omega = 1.25$	linsolve
SOR	$\begin{bmatrix} 2.3131 \\ 3.9077 \\ 4.9695 \\ 5.6528 \\ 6.0780 \\ 6.3342 \\ 6.4838 \\ 6.5687 \\ 6.6153 \\ 6.6401 \\ 6.6521 \\ 6.6548 \\ 6.6460 \\ 6.6150 \\ 6.5366 \\ 6.3635 \\ 6.0170 \\ 5.3803 \\ 4.2965 \\ 2.5744 \end{bmatrix}$	$\begin{bmatrix} 3.1848 \\ 5.6222 \\ 7.4548 \\ 8.8087 \\ 9.7919 \\ 10.4939 \\ 10.9868 \\ 11.3267 \\ 11.5558 \\ 11.7030 \\ 11.7847 \\ 11.8038 \\ 11.7474 \\ 11.5837 \\ 11.2593 \\ 10.6964 \\ 9.7923 \\ 8.4214 \\ 6.4388 \\ 3.6863 \end{bmatrix}$	$\begin{bmatrix} 5.3080 \\ 9.8684 \\ 13.7658 \\ 17.0773 \\ 19.8711 \\ 22.2046 \\ 24.1232 \\ 25.6588 \\ 26.8287 \\ 27.6350 \\ 28.0646 \\ 28.0896 \\ 27.6680 \\ 26.7453 \\ 25.2564 \\ 23.1276 \\ 20.2789 \\ 16.6265 \\ 12.0850 \\ 6.5700 \end{bmatrix}$	$\begin{bmatrix} 6.9394 \\ 13.1475 \\ 18.6717 \\ 23.5486 \\ 27.8041 \\ 31.4540 \\ 34.5036 \\ 36.9482 \\ 38.7745 \\ 39.9606 \\ 40.4780 \\ 40.2922 \\ 39.3642 \\ 37.6518 \\ 35.1106 \\ 31.6955 \\ 27.3614 \\ 22.0647 \\ 15.7640 \\ 8.4207 \end{bmatrix}$	$\begin{bmatrix} 9.8148 \\ 18.4735 \\ 26.2295 \\ 33.0711 \\ 38.9860 \\ 43.9622 \\ 47.9876 \\ 51.0505 \\ 53.1403 \\ 54.2469 \\ 54.3616 \\ 53.4767 \\ 51.5861 \\ 48.6849 \\ 44.7699 \\ 39.8393 \\ 33.8929 \\ 26.9321 \\ 18.9597 \\ 9.9804 \end{bmatrix}$	$\begin{bmatrix} 10.0000 \\ 19.0000 \\ 27.0000 \\ 34.0000 \\ 40.0000 \\ 45.0000 \\ 49.0000 \\ 52.0000 \\ 54.0000 \\ 55.0000 \\ 55.0000 \\ 54.0000 \\ 52.0000 \\ 49.0000 \\ 45.0000 \\ 40.0000 \\ 34.0000 \\ 27.0000 \\ 19.0000 \\ 10.0000 \end{bmatrix}$
$erro_{abs}$	158.8784	141.1205	88.3498	48.8318	2.8557	-
$erro_{rel}$	0.8612	0.7650	0.4789	0.2647	0.0155	-

Com estes resultados, podemos concluir que quanto maior o valor de N, maior será o erro das aproximações (note-se que esta conclusão é coerente com a relação entre N e o condicionamento de A verificado na alínea anterior).

Com o conhecimento do parâmetro ω otimizado, determinado na Alínea c) deste exercício, podemos também concluir, como era de esperar, que quanto mais próximo o valor de ω se encontrar do valor ω_{opt}, menor será o erro da respetiva aproximação.

De seguida, queremos verificar experimentalmente que o método converge se e só se $\omega \in (0, 2)$.

Teorema 1. *O método iterativo $x^{(n+1)} = Cx^{(n)} + d$ converge para a solução do sistema linear $Ax = b$, qualquer que seja $x^{(0)} \in \mathbb{R}^N$, se e só se $r_\sigma(C) < 1$.*

Recorrendo ao Teorema 1, e considerando $C(\omega) = (\omega L + D)^{-1}((1-\omega)D - \omega U)$, foi criada a função *converge* que recebe uma matriz A quadrada e um parâmetro ω, calcula o raio espetral da matriz de iteração C e, caso $r_\sigma(C) < 1$, o programa devolve a mensagem "O modelo converge", ou "O modelo não converge" caso contrário.

Assim, podemos atribuir a A qualquer matriz da Alínea a), escolher qualquer parâmetro ω e verificar, experimentalmente, que converge se e só se $\omega \in (0, 2)$. No ficheiro script $ex2b.m$ podem ser encontrados exemplos do funcionamento desta função que comprovam o pedido.

Por fim, queremos analisar o que acontece para $\omega = 0$ e $\omega = 2$. Recorrendo novamente à função SOR, para cada $N = 5, 10, 15, 20$ são calculadas as aproximações para $\omega = 0$ e $\omega = 2$, com um número máximo de iterações arbitrário. Como $\omega = 0$ e $\omega = 2$ não pertencem ao intervalo $(0, 2)$, já sabemos que o método não irá convergir para uma solução do sistema. Para $\omega = 0$, as soluções são obviamente vetores nulos, uma vez que substituindo $\omega = 0$ na expressão $x^{(n+1)} = (\omega L + D)^{-1}((1-\omega)D - \omega U)x^{(n)} + \omega(\omega L + D)^{-1}b$ obtemos $x^{(n+1)} = x^n$, que é sempre o vetor nulo se $x^{(0)} = 0$.

Para $\omega = 2$ obtemos:

$SOR(A5, b5, 2, 20) = [2; 4; 6; 8; 4]$

$SOR(A10, b10, 2, 20) = [9; 18; 16; 14; 12; 10; 8; 6; 4; 2]$

$SOR(A15, b15, 2, 20) = [4; 8; 12; 16; 20; 24; 28; 32; 36; 40; 44; 48; 36; 24; 12]$

$SOR(A20, b20, 2, 20) = [20; 19; 18; 17; 16; 15; 14; 13; 12; 11; 10; 9; 8; 7; 6; 5; 4; 3; 2; 1]$

É de notar que só se realizaram 20 iterações e que se verificou que, realizando um número maior de iterações, os resultados nalguns casos diferem bastante.

Por exemplo:

$SOR(A10, b10, 2, 100) = [1; 2; 3; 4; 5; 6; 7; 8; 9; 10]$

$SOR(A20, b20, 2, 100) = [16; 32; 48; 64; 80; 75; 70; 65; 60; 55; 50; 45; 40; 35; 30; 25; 20; 15; 10; 5]$

2. c)

Nesta alínea, é pedido para determinarmos o raio espetral da matriz de iteração $C(\omega) = (\omega L + D)^{-1}((1-\omega)D - \omega U)$, para vários valores de $N = 5, 10, 15, 20, \ldots$ e variando o valor de $\omega \in (0, 2)$.

Em primeiro lugar, criámos a função *matrixit* que recebe uma matriz A e um parâmetro ω e devolve a respetiva matriz de iteração.

De seguida, foi definida a função *raioespetral* que recebe uma matriz C e devolve o seu raio espetral, $r_\sigma(C) = \max_{i=1,\ldots,N} |\lambda_i|$, onde λ_i são os valores próprios da matriz.

Assim, para cada N, e fazendo variar $\omega \in (0, 2)$, obtemos vários valores de raios espetrais das várias matrizes iteradoras.

Para $N = 5, 10, 15, 20, 25$ obtemos os seguintes resultados:

$\omega \backslash$N	5	10	15	20	25
0.25	0.9624	0.9885	0.9945	0.9968	0.9979
0.50	0.9140	0.9733	0.9873	0.9926	0.9951
0.75	0.8482	0.9522	0.9771	0.9867	0.9913
1.00	0.7500	0.9206	0.9619	0.9778	0.9855
1.25	0.5603	0.8663	0.9363	0.9629	0.9757
1.50	0.5000	0.7280	0.8804	0.9317	0.9557
1.75	0.7500	0.7500	0.7500	0.7500	0.8756

Nos exemplos acima, foram escolhidos previamente os valores de ω. No entanto, recorrendo ao comando $rand()$, podemos determinar o raio espetral de uma certa matriz para um valor $\omega \in (0, 2)$ aleatório.

Por fim, queremos saber se existe, para N fixo, um valor ótimo de ω, que leve à convergência mais rápida e se este valor depende de N.

Demonstração. Consideremos o método SOR, $x^{(n+1)} = (\omega L + D)^{-1}((1-\omega)D - \omega U)x^{(n)} + \omega(\omega L + D)^{-1}b$. Comecemos por determinar os valores próprios da matriz $C(\omega) = (\omega L + D)^{-1}((1-\omega)D - \omega U)$.

$$
\begin{aligned}
det(C - \lambda I) = 0 \Leftrightarrow det[(D - \omega L)^{-1}((1-\omega)D + \omega U) - \lambda I] = 0 \Leftrightarrow \\
\Leftrightarrow det(D - \omega L)^{-1} det[(1-\omega)D + \omega U - \lambda(D - \omega L)] = 0 \Leftrightarrow \\
\Leftrightarrow det[(1 - \omega - \lambda)D + \omega U + \lambda\omega L] = 0 \Leftrightarrow \\
\Leftrightarrow det[(\frac{(1-\omega\lambda)}{\omega\sqrt{\lambda}}D + \frac{1}{\sqrt{\lambda}}U + \sqrt{\lambda}L)\omega\sqrt{\lambda}] = 0 \Leftrightarrow \\
\Leftrightarrow det[\sqrt{\lambda}D^{-1}L + \frac{1}{\sqrt{\lambda}}D^{-1}U - \frac{(\lambda + \omega - 1)}{\omega\sqrt{\lambda}}I] = 0
\end{aligned}
$$

Quando a matriz é consistentemente ordenada, ou seja, quando os valores próprios de $\alpha L + \frac{1}{\alpha}U$ são independentes de α, os valores próprios de $\sqrt{\lambda}D^{-1}L + \frac{1}{\sqrt{\lambda}}D^{-1}U$ são os mesmos de $D^{-1}(L+U) = C_J$, onde C_J representa a matriz de iteração do sistema linear $Ax = b$ para o método de Jacobi.

Designemos os valores próprios de C_J por μ. Então, os valores próprios de C_{SOR} satisfazem

$$
\begin{aligned}
\mu = \frac{\lambda + \omega - 1}{\omega\sqrt{\lambda}} \Leftrightarrow \mu\omega\sqrt{\lambda} = \lambda + \omega - 1 \Leftrightarrow \\
\Leftrightarrow \mu^2\omega^2\lambda = (\lambda + \omega - 1)^2 \Leftrightarrow \mu^2\omega^2\lambda = \lambda^2 + 2\lambda(\omega - 1) + (\omega - 1)^2 \Leftrightarrow \\
\Leftrightarrow \lambda^2 + \lambda(2\omega - 2 - \mu^2\omega^2) + (\omega - 1)^2 = 0 \Leftrightarrow
\end{aligned}
$$

$$\Leftrightarrow \lambda = 1 - \omega + \frac{\mu^2\omega^2}{2} \pm \mu\omega\sqrt{(1-\omega) + \frac{\mu^2\omega^2}{4}}$$

Para cada μ^2 existem 2 valores possíveis de λ, sendo $|\lambda| = \omega - 1$.
Logo, $r_\sigma(C_{SOR}) = \omega - 1$.

O valor de ω_{opt} deve maximizar a convergência, ou seja, minimizar $r_\sigma(C_{SOR})$, o que significa que queremos $\mu = r_\sigma(C_J)$ máximo:

$$\frac{\mu^2\omega^2}{4} - \omega + 1 = 0 \Leftrightarrow \omega = \frac{2}{1 \mp \sqrt{1-\mu^2}}$$

Assim, podemos concluir que $\omega_{opt} = \frac{2}{1\mp\sqrt{1-r_\sigma(C_J)^2}}$

□

Como podemos analisar pela fórmula demonstrada, o valor ω_{opt} depende de N, já que depende da matriz C_J, a qual varia com N, no entanto, podemos também verificar este facto experimentalmente.

Para a determinação deste valor, que será necessário na Alínea e) deste exercício, criámos a função *wopt* que recebe uma matriz A e determina o valor otimizado de ω recorrendo ao raio espetral da respetiva matriz de iteração pelo método de Jacobi.

$\omega\backslash$N	5	10	15	20	25
ω_{opt}	1.3333	1.5604	1.6735	1.7406	1.7849

Logo, concluímos que ω_{opt} depende de N.

2. d)

Nesta alínea pretendemos verificar, numericamente, que $det(C(\omega)) = (1-\omega)^N$. Temos então:
$det(C(\omega)) = det((\omega L + D)^{-1})$·det((1-$\omega$)$D - \omega U$) = $\frac{1}{2^N}$·(1-ω)^N·$2^N = (1-\omega)^N$
(Notando que $(\omega L + D)^{-1}$ e $((1-\omega)D - \omega U)$ são matrizes triangulares).

Para verificar isto experimentalmente, foi definida a função *detmatrix* que recebe uma matriz A, um parâmetro ω e um número N, e apresenta uma lista onde a primeira componente é o determinante da matriz de iteração $C(\omega)$ e a segunda componente é o valor de $(1-\omega)^N$. Assim, podemos verificar que ambos os valores apresentados são sempre iguais.
Atentendo a este resultado, e sabendo que o determinante de uma matriz é dado pelo produto dos seus valores próprios, podemos fazer a seguinte observação:

Para $\omega \notin (0,2)$ temos que $det(C(\omega)) = (1-\omega)^N \geq 1$, pelo que existe algum valor próprio λ de $C(\omega)$ tal que $\lambda \geq 1$, isto é, $r_\sigma(C(\omega)) \geq 1$ e portanto, pelo Teorema 1 referido na alínea b), o método não converge.

2. e)

Nesta alínea, temos como objetivo aproximar as soluções do sistema $Ax = b$, pelos métodos SOR (com $\omega = \omega_{opt}$), Gauss-Seidel e Jacobi, para $N = 5, 10, 15, 20$, e calcular o erro absoluto em relação às soluções exatas obtidas a partir do comando *linsolve*. Sendo $\tilde{z}$ a solução obtida pelo método e z a solução pelo *linsolve*, o erro absoluto será dado por:

$$erro_{abs} = \|z - \tilde{z}\|$$

Como podemos observar pela expressão do método Gauss-Seidel, $x^{(n+1)} = -(D+L)^{-1}Ux^{(n)} + (D+L)^{-1}b$, este corresponde ao método SOR quando $\omega = 1$.
Para o método de Jacobi, $x^{(n+1)} = -D^{-1}(L+U)x^{(n)} + (D)^{-1}b$, foi criada a função *Jacobi* que funciona de forma idêntica à função *SOR*, não recebendo, apenas, o valor de ω.

Para $N = 5, 10, 15, 20$:

	linsolve	*SOR*	*Gauss − Seidel*	*Jacobi*
solução	$\begin{bmatrix} 2.5000 \\ 4.0000 \\ 4.5000 \\ 4.0000 \\ 2.5000 \end{bmatrix}$	$\begin{bmatrix} 2.4984 \\ 3.9984 \\ 4.4989 \\ 3.9994 \\ 2.4998 \end{bmatrix}$	$\begin{bmatrix} 2.3102 \\ 3.7153 \\ 4.2153 \\ 3.7865 \\ 2.3932 \end{bmatrix}$	$\begin{bmatrix} 1.9463 \\ 3.0508 \\ 3.3926 \\ 3.0508 \\ 1.9463 \end{bmatrix}$
$erro_{abs}$	-	0.0026	0.5051	1.9083

	$linsolve$	SOR	$Gauss - Seidel$	$Jacobi$
solução	$\begin{bmatrix} 5.0000 \\ 9.0000 \\ 12.0000 \\ 14.0000 \\ 15.0000 \\ 15.0000 \\ 14.0000 \\ 12.0000 \\ 9.0000 \\ 5.0000 \end{bmatrix}$	$\begin{bmatrix} 4.7729 \\ 8.5919 \\ 11.5205 \\ 13.5294 \\ 14.5904 \\ 14.6782 \\ 13.7721 \\ 11.8570 \\ 8.9239 \\ 4.9702 \end{bmatrix}$	$\begin{bmatrix} 2.7001 \\ 4.7347 \\ 6.2341 \\ 7.2901 \\ 7.9466 \\ 8.1950 \\ 7.9755 \\ 7.1853 \\ 5.6914 \\ 3.3457 \end{bmatrix}$	$\begin{bmatrix} 2.0840 \\ 3.4131 \\ 4.1914 \\ 4.6143 \\ 4.7891 \\ 4.7891 \\ 4.6143 \\ 4.1914 \\ 3.4131 \\ 2.0840 \end{bmatrix}$
$erro_{abs}$	-	1.0099	16.4611	24.2093

	$linsolve$	SOR	$Gauss - Seidel$	$Jacobi$
solução	$\begin{bmatrix} 7.5000 \\ 14.0000 \\ 19.5000 \\ 24.0000 \\ 27.5000 \\ 30.0000 \\ 31.5000 \\ 32.0000 \\ 31.5000 \\ 30.0000 \\ 27.5000 \\ 24.0000 \\ 19.5000 \\ 14.0000 \\ 7.5000 \end{bmatrix}$	$\begin{bmatrix} 5.4818 \\ 10.4093 \\ 14.8341 \\ 18.8035 \\ 22.3610 \\ 25.5466 \\ 27.4253 \\ 28.4055 \\ 28.4445 \\ 27.5051 \\ 25.5563 \\ 22.5729 \\ 18.5360 \\ 13.4323 \\ 7.2544 \end{bmatrix}$	$\begin{bmatrix} 2.7001 \\ 4.7367 \\ 6.2497 \\ 7.3579 \\ 8.1586 \\ 8.7298 \\ 9.1304 \\ 9.3972 \\ 9.5383 \\ 9.5257 \\ 9.2908 \\ 8.7248 \\ 7.6849 \\ 6.0037 \\ 3.5018 \end{bmatrix}$	$\begin{bmatrix} 2.0840 \\ 3.4141 \\ 4.1953 \\ 4.6328 \\ 4.8438 \\ 4.9453 \\ 4.9805 \\ 4.9922 \\ 4.9805 \\ 4.9453 \\ 4.8438 \\ 4.6328 \\ 4.1953 \\ 3.4141 \\ 2.0840 \end{bmatrix}$
$erro_{abs}$	-	12.8138	63.9149	76.9383

	$linsolve$	SOR	$Gauss-Seidel$	$Jacobi$
solução	$\begin{bmatrix}10.0000\\19.0000\\27.0000\\34.0000\\40.0000\\45.0000\\49.0000\\52.0000\\54.0000\\55.0000\\55.0000\\54.0000\\52.0000\\49.0000\\45.0000\\40.0000\\34.0000\\27.0000\\19.0000\\10.0000\end{bmatrix}$	$\begin{bmatrix}6.0121\\11.5094\\16.5332\\21.1218\\25.3107\\29.1329\\32.6186\\35.7960\\38.6909\\41.3274\\43.7273\\44.0930\\43.4802\\41.8518\\39.1761\\35.4266\\30.5823\\24.6268\\17.5486\\9.3407\end{bmatrix}$	$\begin{bmatrix}2.7001\\4.7367\\6.2497\\7.3579\\8.1586\\8.7298\\9.1323\\9.4128\\9.6061\\9.7380\\9.8272\\9.8849\\9.9108\\9.8846\\9.7571\\9.4441\\8.8254\\7.7496\\6.0433\\3.5216\end{bmatrix}$	$\begin{bmatrix}2.0840\\3.4141\\4.1953\\4.6328\\4.8438\\4.9453\\4.9814\\4.9961\\4.9990\\5.0000\\5.0000\\4.9990\\4.9961\\4.9814\\4.9453\\4.8438\\4.6328\\4.1953\\3.4141\\2.0840\end{bmatrix}$
$erro_{abs}$	–	46.8058	147.7404	164.7817

Através da observação dos resultados acima, podemos concluir que o melhor método iterativo é o método SOR, já que, para todos os valores de N, apresenta um erro absoluto em relação ao valor exato obtido com o comando $linsolve$ muito inferior aos erros dos outros dois métodos, que apresentam, de maneira geral, erros absolutos próximos, apesar do método $Gauss-Seidel$, uma particularidade do método SOR, ter tendência a ser um pouco melhor que o método $Jacobi$.

Por fim, podemos ainda concluir que o aumento do valor de N leva a um aumento do erro absoluto das aproximações.