Esercizi Elaborato (versione 2022-04-27)

N.B.: curare attentamente la stesura delle function Matlab, che devono essere allegate all'elaborato in un file `.zip`

Esercizio 1. Verificare che,

$$\frac{-f(x+2h)+8f(x+h)-8f(x-h)+f(x-2h)}{12h} = f'(x) + O(h^4).$$

Esecizio 2. Calcolare, motivandone i passaggi, la precisione di macchina della doppia precisione dello standard IEEE. Confrontare questa quantità con quanto ritornato dalla variabile `eps` di Matlab, commentando a riguardo.

Esercizio 3. Eseguire il seguente *script* Matlab:

```
format long e
(1+(1e-14-1))*1e14
```

Spiegare i risultati ottenuti.

Esercizio 4. Scrivere una *function* Matlab, `radice(x)` che, avendo in ingresso un numero `x` non negativo, ne calcoli la radice quadrata utilizzando solo operazioni algebriche elementari, con la massima precisione possibile. Confrontare con la *function* `sqrt` di Matlab per 20 valori di `x`, equispaziati logaritmicamente nell'intervallo `[1e-10,1e10]`, evidenziando che si è ottenuta la massima precisione possibile.

Esercizio 5. Scrivere *function* Matlab distinte che implementino efficientemente i seguenti metodi per la ricerca degli zeri di una funzione $f(x)$:

- il metodo di Newton;
- il metodo delle secanti;
- il *metodo di Steffensen*:

$$x_{n+1} = x_n - \frac{f(x_n)^2}{f(x_n + f(x_n)) - f(x_n)}, \qquad n = 0, 1, \ldots.$$

Per tutti i metodi, utilizzare come criterio di arresto

$$|x_{n+1} - x_n| \le tol \cdot (1 + |x_n|),$$

essendo *tol* una opportuna tolleranza specificata in ingresso. Curare particolarmente la robustezza del codice.

Esercizio 6. Utilizzare le *function* del precedente esercizio per determinare una approssimazione della radice della funzione

$$f(x) = x - \cos\left(\frac{\pi}{2}x\right),$$

per $tol = 10^{-3}, 10^{-6}, 10^{-9}, 10^{-12}$, partendo da $x_0 = 1$ (e $x_1 = 0.99$ per il metodo delle secanti). Tabulare i risultati, in modo da confrontare le iterazioni richieste da ciascun metodo. Commentare il relativo costo computazionale, in termini di valutazioni funzionali richieste.

Esercizio 7. Utilizzare le *function* del precedente Esercizio 5 per determinare una approssimazione della radice della funzione

$$f(x) = \left[x - \cos\left(\frac{\pi}{2}x\right)\right]^3,$$

per $tol = 10^{-3}, 10^{-6}, 10^{-9}, 10^{-12}$, partendo da $x_0 = 1$ (e $x_1 = 0.99$ per il metodo delle secanti). Tabulare i risultati, in modo da confrontare le iterazioni richieste da ciascun metodo. Commentare i risultati ottenuti.

Esercizio 8. Scrivere una *function* Matlab,

```
function x = mialu(A,b)
```

che, data in ingresso una matrice A ed un vettore $\boldsymbol{b}$, calcoli la soluzione del sistema lineare $A\boldsymbol{x} = \boldsymbol{b}$ con il metodo di fattorizzazione LU con *pivoting* parziale. Curare particolarmente la scrittura e l'efficienza della *function*, e validarla su due esempi non banali, generati casualmente, di cui sia nota la soluzione.

Esercizio 9. Scrivere una *function* Matlab,

```
function x = mialdl(A,b)
```

che, dati in ingresso una matrice sdp A ed un vettore $\boldsymbol{b}$, calcoli la soluzione del corrispondente sistema lineare utilizzando la fattorizzazione $LDL^\top$. Curare particolarmente la scrittura e l'efficienza della *function*, e validarla su due esempi non banali, generati casualmente, di cui sia nota la soluzione.

Esercizio 10. Data la *function* Matlab

```
function [A,b] = linsis(n,k,simme)
%
%  [A,b] = linsis(n,k,simme)  Crea una matrice A nxn ed un termine noto b,
%                             in modo che la soluzione del sistema lineare
%                             A*x=b sia x = [1,2,...,n]'.
%                             k      è un parametro ausiliario.
%                             simme, se specificato, crea una matrice
%                                    simmetrica e definita positiva.
%
sigma = 10^(-2*(1-k))/n;
rng(0);
[q1,r1] = qr(rand(n));
```

```
if nargin==3
    q2 = q1';
else
    [q2,r1] = qr(rand(n));
end
A = q1*diag([sigma 2/n:1/n:1])*q2;
x = [1:n]';
b = A*x;
return
```

che crea sistemi lineari casuali con soluzione nota, risolvere, utilizzando la *function* `mialu`, i sistemi lineari generati da `[A,b]=linsis(10,1)` e `[A,b]=linsis(10,10)`. Commentare l'accuratezza dei risultati ottenuti, dandone spiegazione esaustiva.

Esercizio 11. Risolvere, utilizzando la *function* `mialdlt`, i sistemi lineari generati da `[A,b]=linsis(10,1,1)` e `[A,b]=linsis(10,10,1)`. Commentare l'accuratezza dei risultati ottenuti, dandone spiegazione esaustiva.

Esercizio 12. Scrivere una *function* Matlab,

```
function [x,nr] = miaqr(A,b)
```

che, data in ingresso la matrice A $m \times n$, con $m \geq n = \mathrm{rank}(A)$, ed un vettore $\boldsymbol{b}$ di lunghezza m, calcoli la soluzione del sistema lineare $A\boldsymbol{x} = \boldsymbol{b}$ nel senso dei minimi quadrati e, inoltre, la norma, nr, del corrispondente vettore residuo. Curare particolarmente la scrittura e l'efficienza della *function*. Validare la *function* `miaqr` su due esempi non banali, generati casualmente, confrontando la soluzione ottenuta con quella calcolata con l'operatore Matlab \

Esercizio 13. Utilizzare la *function* `miaqr` per risolvere, nel senso dei minimi quadrati, i sistemi lineari sovradeterminati

```
A x = b,        (D*A)x = (D*b)
```

definiti dai seguenti dati:

```
A = [ 1 3 2; 3 5 4; 5 7 6; 3 6 4; 1 4 2 ];
b = [ 15  28  41 33  22 ]';
D = diag(1:5);
```

Commentare i risultati ottenuti.

Esercizio 14. Scrivere una *function* Matlab,

```
[x,nit] = newton(fun,jacobian,x0,tol,maxit)
```

che implementi efficientemente il metodo di Newton per risolvere sistemi di equazioni nonlineari. Curare particolarmente il criterio di arresto, che deve essere analogo a quello usato nel caso scalare. La seconda variabile, se specificata, ritorna il numero di iterazioni eseguite. Prevedere opportuni valori di *default* per gli ultimi due parametri di ingresso.

Esercizio 15. Usare la *function* del precedente esercizio per risolvere, a partire dal vettore iniziale nullo, i seguenti sistemi nonlineari, utilizzando tolleranze `tol = 1e-3, 1e-8, 1e-13`:

$$f_1(\boldsymbol{x}) = \begin{pmatrix} (x_1^2+1)(x_2-2) \\ \exp(x_1-1)+\exp(x_2-2)-2 \end{pmatrix}, \qquad f_2(\boldsymbol{x}) = \begin{pmatrix} x_1 - x_2x_3 \\ \exp(x_1+x_2+x_3-3)-x_2 \\ x_1+x_2+2x_3-4 \end{pmatrix}.$$

Tabulare i risultati ottenuti, commentandone l'accuratezza.

Esercizio 16. Costruire una function, `lagrange.m`, avente la <u>stessa sintassi</u> della function `spline` di Matlab, che implementi, in modo vettoriale, la forma di Lagrange del polinomio interpolante una funzione.

Esercizio 17. Costruire una function, `newton.m`, avente la <u>stessa sintassi</u> della function `spline` di Matlab, che implementi, in modo vettoriale, la forma di Newton del polinomio interpolante una funzione.

Esercizio 18. Costruire una function, `hermite.m`, avente una sintassi <u>analoga</u> alla function `spline` di Matlab (in pratica, con un parametro di ingresso in più per i valori delle derivate), che implementi, in modo vettoriale, il polinomio interpolante di Hermite.

Esercizio 19. Costruire una function Matlab che, specificato in ingresso il grado n del polinomio interpolante, e gli estremi dell'intervallo $[a, b]$, calcoli le corrispondenti ascisse di Chebyshev.

Esercizio 20. Costruire una function, `spline0.m`, avente la <u>stessa sintassi</u> della function `spline` di Matlab, che implementi la spline cubica naturale interpolante una funzione.

Esercizio 21. Costruire una tabella in cui viene riportato, al crescere di n, il massimo errore di interpolazione ottenuto approssimando la funzione:

$$f(x) = \frac{1}{2(2x^2-2x+1)}$$

sulle ascisse $x_0 < x_1 \cdots < x_n$:

- equidistanti in $[-2, 3]$,
- di Chebyshev per lo stesso intervallo,

utilizzando le function degli Esercizi 16–18 e 20, e la function `spline` di Matlab. Considerare $n = 4, 8, 16, \ldots, 40$ e stimare l'errore di interpolazione su 10001 punti equidistanti nell'intervallo $[x_0, x_n]$.

Esercizio 22. Tabulare il massimo errore di approssimazione (stimato su 10001 punti equidistanti in $[0, 1]$) ottenuto approssimando le funzioni

$$\sin(2\pi x) \qquad \text{e} \qquad \cos(2\pi x)$$

mediante le function `spline0` e `spline`, interpolanti su $n+1$ punti equidistanti in $[0, 1]$, per $n = 5, 10, 15, 20, \ldots, 50$. Commentare i risultati ottenuti.

Esercizio 23. Sia assegnata la seguente perturbazione della funzione $f(x) = \sin(\pi x^2)$:

$$\tilde{f}(\mathtt{x}) = f(\mathtt{x}) + 10^{-1}\,\mathtt{rand(size(x))},$$

in cui `rand` è la function built-in di Matlab. Calcolare polinomio di approssimazione ai minimi quadrati di grado m, $p(x)$, sui dati $(x_i, \tilde{f}(x_i))$, $i = 0, \ldots, n$, con:

$$x_i = i/n, \qquad n = 10^4.$$

Graficare (in formato `semilogy`) l'errore di approssimazione $\|f - p\|$ (stimato come il massimo errore sui punti x_i), relativo all'intervallo $[0, 1]$, rispetto ad m, per $m = 1, 2, \ldots, 15$. Commentare i risultati ottenuti.

Esercizio 24. Costruire una function Matlab che, dato in input n, restituisca i pesi della quadratura della formula di Newton-Cotes di grado n. Tabulare, quindi, i pesi delle formule di grado 1, 2,. . . , 7 e 9 (come numeri razionali).

Esercizio 25. Utilizzare le formule tabulate nel precedente esercizio per calcolare le approssimazioni dell'integrale

$$I(f) = \int_0^1 e^{3x}\mathrm{d}x,$$

tabulando (in modo significativo) il corrispondente errore di quadratura (risolvere a mano l'integrale).

Esercizio 26. Scrivere una function Matlab,

```
[If,err,nfeval] = composita( fun, a, b, n, tol)
```

in cui:

- `fun` è l'identificatore di una function che calcoli (in modo vettoriale) la funzione integranda,
- `a` e `b` sono gli estremi dell'intervallo di integrazione,
- `n` è il grado di una formula di Newton-Cotes base,
- `tol` è l'accuratezza richiesta,

che calcoli, fornendo la stima `err` dell'errore di quadratura, l'approssimazione `If` dell'integrale, raddoppiando il numero di punti ed usando la formula composita corrispondente per stimare l'errore di quadratura, fino a soddisfare il requisito di accuratezza richiesto. In uscita è anche il numero totale di valutazioni funzionali effettuate, `nfeval`.
N.B.: evitare di effettuare valutazioni di funzione ridondanti.

Esercizio 27. Tabulare il numero di valutazioni di funzione richieste per calcolare, mediante la function del precedente esercizio, l'approssimazione dell'integrale

$$I(f) = \int_0^1 \sin\left(\frac{1}{0.1 + x}\right)\mathrm{d}x$$

utilizzando le formule di Newton-Cotes di grado $n = 1, \ldots, 7$, e 9, e tolleranze
$tol = 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}$.

Esercizio 28. Scrivere una function che implementi la formula adattiva dei trapezi.
N.B.: evitare di effettuare valutazioni di funzione ridondanti.

Esercizio 29. Scrivere una function che implementi la formula adattiva di Simpson.
N.B.: evitare di effettuare valutazioni di funzione ridondanti.

Esercizio 30. Tabulare il numero di valutazioni di funzione richieste dalle function degli Esercizi 29 e 30 per approssimare l'integrale

$$I(f) = \int_0^1 \sin\left(\frac{1}{0.01 + x}\right) \mathrm{d}x$$

con tolleranze $tol = 10^{-2}, 10^{-3}, 10^{-4}, 10^{-5}, 10^{-6}$.