Esercizi Elaborato (versione 2023-05-01)

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

Esercizio 1. Verificare che:

$$-\frac{1}{4}f(x-h)-\frac{5}{6}f(x)+\frac{3}{2}f(x+h)-\frac{1}{2}f(x+2h)+\frac{1}{12}f(x+3h)=hf'(x)+O(h^5).$$

Esecizio 2. Matlab utilizza la doppia precisione IEEE. Stabilire, pertanto, il nesso tra la variabile `eps` e la precisione di macchina di questa aritmetica.

Esercizio 3. Spiegare il fenomeno della *cancellazione numerica*. Fare un esempio che la illutri, spiegandone i dettagli.

Esercizio 4. Scrivere una *function* Matlab, `radice(x)` che, avendo in ingresso un numero `x` non negativo, calcoli $\sqrt[6]{\mathtt{x}}$ utilizzando solo operazioni algebriche elementari, con la massima precisione possibile. Confrontare con il risultato fornito da `x^(1/6)` per 20 valori di `x`, equispaziati logaritmicamente nell'intervallo `[1e-10,1e10]`, tabulando i risultati in modo che si possa <u>verificare</u> che si è ottenuta la massima precisione possibile.

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

$$f(x)=x-\cos(x),$$

per $tol=10^{-3},10^{-6},10^{-9},10^{-12}$, partendo da $x_0=0$ (e $x_1=0.1$ per il metodo delle secanti). Tabulare i risultati, in modo da confrontare il costo computazionale di ciascun metodo.

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

$$f(x)=\left[x-\cos(x)\right]^5,$$

per $tol=10^{-3},10^{-6},10^{-9},10^{-12}$, partendo da $x_0=0$ (e $x_1=0.1$ per il metodo delle secanti). Confrontare con i risultati ottenuti utilizzando il metodo di Newton modificato. Tabulare i risultati, in modo da confrontare il costo computazionale e l'accuratezza di 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. Scrivere una *function* Matlab,

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

che, data in ingresso la matrice A $m \times n$, con $m \geq n = \text{rank}(A)$, 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 11. Data la *function* Matlab

```
function [A1,A2,b1,b2] = linsis1(n,simme)
%
%
rng(0);
[q1,r1] = qr(rand(n));
if nargin==2
    q2 = q1';
else
    [q2,r1] = qr(rand(n));
end
A1 = q1*diag([1 2/n:1/n:1])*q2;
A2 = q1*diag([1e-10 2/n:1/n:1])*q2;
b1 = sum(A1,2);
b2 = sum(A2,2);
return
```

che crea sistemi lineari casuali di dimensione n con soluzione nota,

$$A_1x = b_1, \qquad A_2x = b_2, \qquad x = (1, \ldots, 1)^\top \in \mathbb{R}^n,$$

risolvere, utilizzando la *function* `mialu`, i sistemi lineari generati da `[A1,A2,b1,b2]=linsis(5)`. Commentare l'accuratezza dei risultati ottenuti, dandone spiegazione esaustiva.

Esercizio 12. Risolvere, utilizzando la *function* `mialdlt`, i sistemi lineari generati da `[A1,A2,b1,b2]=linsis(5,1)`. Commentare l'accuratezza dei risultati ottenuti, dandone spiegazione esaustiva.

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),        (D1*A)x = (D1*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);
D1 = diag(pi*[1 1 1 1 1]).
```

Calcolare le corrispondenti soluzioni e residui, e 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, il sistema nonlineare derivante dalla determinazione del punto stazionario della funzione:

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top Q\boldsymbol{x} - \boldsymbol{e}^\top \left[\sin\left(\frac{\pi}{2}\boldsymbol{x}\right) + \boldsymbol{x}\right], \qquad \boldsymbol{e} = \frac{1}{100}\begin{pmatrix} 1 \\ 2 \\ \vdots \\ 100 \end{pmatrix} \in \mathbb{R}^{100},$$

$$Q = \begin{pmatrix} 2 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & 2 \end{pmatrix} \in \mathbb{R}^{100\times 100}, \qquad \sin\left(\frac{\pi}{2}\boldsymbol{x}\right) = \begin{pmatrix} \sin\left(\frac{\pi}{2}x_1\right) \\ \sin\left(\frac{\pi}{2}x_2\right) \\ \vdots \\ \sin\left(\frac{\pi}{2}x_{100}\right) \end{pmatrix}.$$

utilizzando tolleranze `tol = 1e-3, 1e-8, 1e-13`. Graficare la soluzione e tabulare in modo conveniente i risultati ottenuti.

Esercizio 16. Costruire una function, `lagrange.m`, avente la <u>stessa sintassi</u> della function `spline` di Matlab, che implementi, in modo <u>vettoriale</u>, la forma di Lagrange del polinomio interpolante una funzione.
N.B.: il risultato dovrà avere <u>le stesse dimensioni</u> del dato di ingresso; questo vale anche per gli esercizi a seguire.

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

Esercizio 18. Costruire una function, `hermite.m`, avente sintassi

```
yy = hermite( xi, fi, f1i, xx )
```

che implementi, in modo <u>vettoriale</u>, 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 Matlab, con sintassi

```
ll = lebesgue( a, b, nn, type ),
```

che approssimi la costante di Lebesgue per l'interpolazione polinomiale sull'intervallo `[a,b]`, per i polinomi di grado specificato nel vettore `nn`, utilizzando ascisse equidistanti, se `type=0`, o di Chebyshev, se `type=1` (utilizzare 10001 punti equispaziati nell'intervallo `[a,b]` per ottenere ciascuna componente di `ll`). Graficare i risultati ottenuti, per `nn=1:100`, utilizzando `[a,b]=[0,1]` e `[a,b]=[-3,7]`. Giustificare i risultati ottenuti.

Esercizio 21. Utilizzando le function dei precedenti esercizi, graficare (in semilogy) l'andamento errore di interpolazione (utilizzare 10001 punti equispaziati nell'intervallo per ottenerne la stima) per la funzione di Runge,

$$f(x) = \frac{1}{1+x^2}, \qquad x \in [-5, 5],$$

utilizzando sia le ascisse equidistanti che di Chebyshev, per i polinomi interpolanti di grado `nn=2:2:100`. Graficare l'errore di interpolazione anche per i polinomi interpolanti di Hermite di grado `nn=3:2:99`, sia utilizzando ascisse equidistanti che ascisse di Chebyshev, nell'intervallo considerato.

Esercizio 22. Costruire una function, `myspline.m`, avente sintassi

```
yy = myspline( xi, fi, xx, type )
```

dove `type=0` calcola la *spline* cubica interpolante naturale i punti `(xi,fi)`, e `type~=0` calcola quella *not-a-knot* (default).

Esercizio 23. Graficare, utilizzando il formato semilogy, l'errore di approssimazione utilizzando le *spline* interpolanti naturale e *not-a-knot* per approssimare la funzione di Runge sull'intervallo [-5,5], utilizzando una partizione $\Delta = \{-5 = x_0 < \cdots < x_n = 5\}$, con ascisse equidistanti e `n=4:4:400`. Utilizzare 10001 punti equispaziati nell'intervallo $[-5, 5]$ per ottenere la stima dell'errore.

Esercizio 24. È noto che un fenomeno fisico evolve come $y = x^n$, con n incognito. Il file `data.mat`, reperibile a:
`https://drive.google.com/file/d/14u2Pjnl_BZN1GWEFCZd0tqKCObGGI-M5/view?usp=share_link`
contiene 1000 coppie di dati (x_i, y_i), in cui la seconda componente è affetta da un errore con distribuzione Gaussiana a media nulla e varianza "piccola". Utilizzando un opportuno polinomio di approssimazione ai minimi quadrati, stimare il grado n. Argomentare il procedimento seguito,

graficando la norma del residuo rispetto a valori crescenti di n. È richiesto il codice Matlab dell'algoritmo che avrete implementato (potete utilizzare, se lo ritenete opportuno, la function `polyfit` di Matlab).

Esercizio 25. 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 <u>numeri razionali</u>).

Esercizio 26. Scrivere una function Matlab,

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

che implementi la formula composita di Newton-Cotes di grado `k` su `n+1` ascisse equidistanti, con `n` multiplo pari di `k`, in cui:

- `fun` è la funzione integranda (che accetta input vettoriali);
- `[a,b]` è l'intervallo di integrazione;
- `k, n` come su descritti;
- `If` è l'approssimazione dell'integrale ottenuta;
- `err` è la stima dell'errore di quadratura.

Esercizio 27. Utilizzare la function `composita` per ottenere l'approssimazione dell'integrale

$$\int_0^1 \left(\sum_{i=1}^{5} i \cos 2\pi i x - \mathrm{e}^i \sin 2(\pi i + 0.1)x \right) \mathrm{d}x,$$

con le formule composite di Newton-Cotes di grado `k=1,2,3,4,5,6`. Per tutte, utilizzare `n=12`.

Esercizio 28. Implementare la formula composita adattativa di Simpson.

Esercizio 29. Implementare la formula composita adattativa di Newton-Cotes di grado `k=4`.

Esercizio 30. Confrontare le formule adattative degli ultimi due esercizi, tabulando il numero di valutazioni funzionali effettuate, rispetto alla tolleranza `tol = 1e-2, 1e-3, ..., 1e-9`, per ottenere l'approssimazione dell'integrale

$$\int_{10^{-5}}^{1} x^{-1} \cos(\log x^{-1}) \mathrm{d}x \equiv \sin \log 10^5.$$

Costruire un'altra tabella, in cui si tabula l'errore vero (essendo l'integrale noto, in questo caso) rispetto a `tol`.