Esercizi Elaborato (versione 2024-04-06)

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

Esecizio 1. Dimostrare che:

$$\frac{25f(x) - 48f(x-h) + 36f(x-2h) - 16f(x-3h) + 3f(x-4h)}{12h} = f'(x) + O(h^4).$$

Esercizio 2. La funzione

$$f(x) = 1 + x^2 + \frac{\log(|3(1-x)+1|)}{80}, \qquad x \in [1, 5/3],$$

ha un asintoto in $x = 4/3$, in cui tende a $-\infty$. Graficarla in Matlab, utilizzando

$$\boldsymbol{x} = \texttt{linspace}(1, 5/3, 100001)$$

(in modo che il floating di 4/3 sia contenuto in $\boldsymbol{x}$) e vedere dove si ottiene il minimo. Commentare i risultati ottenuti.

Esercizio 3. Spiegare in modo esaustivo il fenomeno della *cancellazione numerica*. Fare un esempio che la illustri, spiegandone i dettagli.

Esercizio 4. Scrivere una *function* Matlab che implementi in modo efficiente il metodo di bisezione.

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

Esercizio 6. Utilizzare le *function* dei precedenti esercizi per determinare una approssimazione della radice della funzione

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

per $tol = 10^{-3}, 10^{-6}, 10^{-9}, 10^{-12}$, partendo da $x_0 = 1$ (e $x_1 = 0.9$ per il metodo delle secanti). Per il metodo di bisezione, usare l'intervallo di confidenza iniziale $[-0.1, 1]$. Tabulare i risultati, in modo da confrontare il costo computazionale di ciascun metodo.

Esercizio 7. Applicare gli stessi metodi e dati del precedente esercizio, insieme al metodo di Newton modificato, per la funzione

$$f(x) = e^x - \cos x + \sin x - x(x+2).$$

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 un congruo numero di esempi significativi, che evidenzino tutti i suoi possibili *output*.

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 un congruo numero di esempi significativi, che evidenzino tutti i suoi possibili *output*.

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*, e validarla su un congruo numero di esempi significativi, che evidenzino tutti i suoi possibili *output*.

Esercizio 11. Risolvere i sistemi lineari, di dimensione n,

$$A_n\boldsymbol{x}_n = \boldsymbol{b}_n, \qquad n = 1, \ldots, 15,$$

in cui

$$A_n = \begin{pmatrix} 1 & 1 & \ldots & \ldots & 1 \\ 10 & \ddots & \ddots & & \vdots \\ 10^2 & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 1 & 1 \\ 10^{n-1} & \ldots & 10^2 & 10 & 1 \end{pmatrix} \in \mathbb{R}^{n\times n}, \qquad \boldsymbol{b}_n = \begin{pmatrix} n-1+\frac{10^1-1}{9} \\ n-2+\frac{10^2-1}{9} \\ n-3+\frac{10^3-1}{9} \\ \vdots \\ 0+\frac{10^n-1}{9} \end{pmatrix} \in \mathbb{R}^n,$$

la cui soluzione è il vettore $\boldsymbol{x}_n = (1, \ldots, 1)^\top \in \mathbb{R}^n$, utilizzando la *function* `mialu`. Tabulare e commentare l'accuratezza dei risultati ottenuti, dandone spiegazione esaustiva.

Esercizio 12. Fattorizzare, utilizzando la *function* `mialdlt`, le matrici sdp

$$A_n = \begin{pmatrix} n & -1 & \ldots & -1 \\ -1 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & -1 \\ -1 & \ldots & -1 & n \end{pmatrix} \in \mathbb{R}^{n\times n}, \qquad n = 1, \ldots, 100.$$

Graficare, in un unico grafico, gli elementi diagonali del fattore D, rispetto all'indice diagonale.

Esercizio 13. Utilizzare la *function* `miaqr` per risolvere, nel senso dei minimi quadrati, il sistema lineare sovradeterminato

$$A\boldsymbol{x} = \boldsymbol{b},$$

in cui

$$A = \begin{pmatrix} 7 & 2 & 1 \\ 8 & 7 & 8 \\ 7 & 0 & 7 \\ 4 & 3 & 3 \\ 7 & 0 & 10 \end{pmatrix}, \qquad \boldsymbol{b} = \begin{pmatrix} 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{pmatrix},$$

dove viene minimizzata la seguente norma *pesata* del residuo $\boldsymbol{r} = (r_1, \ldots, r_5)^T$:

$$\rho_\omega^2 := \sum_{i=1}^{5} \omega_i r_i^2,$$

con

$$\omega_1 = \omega_2 = 0.5, \quad \omega_3 = .75, \quad \omega_4 = \omega_5 = 0.25.$$

Dettagliare l'intero procedimento, calcolando, in uscita, anche ρ_ω.

Esercizio 14. Scrivere una *function* Matlab,

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

che implementi efficientemente il metodo di Newton per risolvere sistemi di equazioni nonlineari. Curare particolarmente il criterio di arresto. La seconda variabile, se specificata, ritorna il numero di iterazioni eseguite. Prevedere opportuni valori di *default* per gli ultimi due parametri di ingresso (rispettivamente, la tolleranza per il criterio di arresto, ed il massimo numero di iterazioni). La function `fun` deve avere sintassi: `[f,jacobian]=fun(x)`, se il sistema da risolvere è `f(x)=0`.

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:

$$f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x}^\top Q\boldsymbol{x} + \boldsymbol{e}^\top \left[\cos(\alpha\boldsymbol{x}) + \beta\exp(-\boldsymbol{x})\right], \qquad \boldsymbol{e} = (1, \ldots, 1)^\top \in \mathbb{R}^{50},$$

$$Q = \begin{pmatrix} 4 & 1 & & \\ 1 & \ddots & \ddots & \\ & \ddots & \ddots & 1 \\ & & 1 & 4 \end{pmatrix} \in \mathbb{R}^{50\times 50}, \qquad \alpha = 2, \quad \beta = -1.1,$$

utilizzando tolleranze `tol = 1e-3, 1e-8, 1e-13` (le *function* cos e exp sono da intendersi in modo vettoriale). Graficare la soluzione e tabulare in modo conveniente i risultati ottenuti.

Esercizio 16. Costruire una function, `lagrange.m`, avente la stessa sintassi della function `spline` di Matlab, che implementi, in modo vettoriale, la forma di Lagrange del polinomio interpolante una funzione.
N.B.: il risultato dovrà avere le stesse dimensioni del dato di ingresso; questo vale anche per gli esercizi 17, 18 e 24.

Esercizio 17. Costruire una function, `newton.m`, avente la stessa sintassi 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 sintassi

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

che implementi, in modo vettoriale, il polinomio interpolante di Hermite.

Esercizio 19. Si consideri la seguente base di Newton,

$$\omega_i(x) = \prod_{j=0}^{i-1}(x - x_j), \qquad i = 0, \ldots, n,$$

con $x_0, \ldots, x_n$ ascisse date (non necessariamente distinte tra loro), ed un polinomio rappresentato rispetto a tale base,

$$p(x) = \sum_{i=0}^{n} a_i \omega_i(x).$$

Derivare una modifica dell' algoritmo di Horner per calcolarne efficientemente la derivata prima.

Esercizio 20. Utilizzando le function degli esercizi 18 e 19, calcolare il polinomio interpolante di Hermite la funzione $f(x) = \exp(x/2 + \exp(-x))$ sulle ascisse equidistanti $\{0, 2.5, 5\}$. Graficare il grafico della funzione interpolanda e del polinomio interpolante nell'intervallo $[0, 5]$, e quello della derivata prima della funzione interpolanda, e della derivata prima del polinomio interpolante, verificando graficamente le condizioni di interpolazione per entrambi.

Esercizio 21. 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 22. 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 opportunamente i risultati ottenuti, per `nn=1:100`, utilizzando `[a,b]=[0,1]` e `[a,b]=[-5,8]`. Commentare i risultati ottenuti.

Esercizio 23. Utilizzando le function degli esercizi 16 e 17, 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 le ascisse di Chebyshev, per i polinomi interpolanti di grado `n=2:2:100`. Commentare i risultati ottenuti.

Esercizio 24. Costruire una function, `spline0.m`, avente la stessa sintassi della function `spline` di Matlab, che calcoli la *spline* cubica interpolante naturale i punti `(xi,fi)`.

Esercizio 25. Graficare, utilizzando il formato loglog, l'errore di approssimazione utilizzando le *spline* interpolanti naturale e *not-a-knot* per approssimare la funzione di Runge sull'intervallo $[-10, 10]$, utilizzando una partizione uniforme

$$\Delta = \left\{ x_i = -10 + i\frac{20}{n},\ i = 0, \dots, n \right\}, \qquad n = \texttt{4 : 4 : 800},$$

rispetto alla distanza $h = 20/n$ tra le ascisse. Utilizzare 10001 punti equispaziati nell'intervallo $[-10, 10]$ per ottenere la stima dell'errore. Che tipo di decrescita si osserva?

Esercizio 26. Graficare, utilizzando il formato loglog, l'errore di approssimazione utilizzando le *spline* interpolanti naturale e *not-a-knot* per approssimare la funzione di Runge sull'intervallo [0,10], utilizzando una partizione uniforme

$$\Delta = \left\{ x_i = i\frac{20}{n},\ i = 0, \dots, n \right\}, \qquad n = \texttt{4 : 4 : 800},$$

rispetto alla distanza $h = 10/n$ tra le ascisse. Utilizzare 10001 punti equispaziati nell'intervallo $[0, 10]$ per ottenere la stima dell'errore. Che tipo di decrescita si osserva? Confrontare e discutere i risultati ottenuti, rispetto a quelli del precedente esercizio.

Esercizio 27. Calcolare i coefficienti del polinomio di approssimazione ai minimi quadrati di grado 3 per i seguenti dati:

```
>> rng(0)
>> xi=linspace(0,2*pi,101);
>> yi=sin(xi)+rand(size(xi))*.05;
```

Graficare convenientemente i risultati ottenuti.

Esercizio 28. 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 29. 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.

Le valutazioni funzionali devono essere fatte tutte insieme in modo vettoriale, senza ridondanze.

Esercizio 30. Calcolare l'espressione del seguente integrale:

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

Utilizzare la function del precedente esercizio per ottenere un'approssimazione dell'integrale per i valori $k = 1, 2, 3, 6$, e $n = 12$. Tabulare i risultati ottenuti, confrontando l'errore stimato con quello vero.