Metodi Numerici per il Calcolo

# Esercitazione 4: Script e Function, Grafica 2D e Funzioni Polinomiali

A.A.2021/22

Scaricare da Virtuale l'archivio matlab_mnc2021_4.zip e scompattarlo nella propria directory matlab_mnc2021. Verrà creata una cartella con lo stesso nome contenente alcuni semplici script e function Matlab/Octave. Si svolga la seguente esercitazione che ha come obiettivo approfondire la propria conoscenza dell'ambiente Matlab capendo, usando e modificando piccoli script e function sulla rappresentazione grafica di dati e funzioni polinomiali.

**A. Function plot()**
Dopo aver visto le potenzialità della grafica Matlab, realizzare lo script `sgrafici1.m` che visualizzi in una stessa `figure` le due funzioni

$$y = \cosh(x), \qquad y = 0.5\exp(x), \qquad\qquad \text{con} \qquad x \in [-2, 2],$$

utilizzando tipi di linee e/o colori differenti, una legenda, un titolo, etichette sugli assi coordinati e una griglia.

**B. Ancora sulla Function plot()**
Realizzare lo script `sgrafici2.m` per rappresentare le seguenti funzioni, ciascuna in una differente `figure`:

$$\begin{aligned} &y = x^3 - 4x && x \in [-3, 3], \\ &y = 0.2x^4 + x^3 - 0.4x^2 - 3x + 1 && x \in [-6, 6], \\ &y = 3\cos(2x) - 2\cos(x) && x \in [0, 2\pi], \\ &y = \frac{\sin(2x)}{x} && x \in [-6\pi, 6\pi]. \end{aligned}$$

**C. Valutazione polinomio e derivata prima**
Si realizzi uno script che utlizzi le function Matlab `polyval` e `polyder` per valutare un polinomio e la sua derivata prima in corrispondenza di un vettore di punti. Lo script si chiami `spoly_eval_der.m`, consideri i polinomi dell'esercizio precedente e li rappresenti graficamente insieme alle loro derivate prime.

**D. Errore Algoritmico nella Valutazione polinomiale**
Completare la function `poly_eval.m` che implementa l'algoritmo di Horner per valutare un polinomio in corrispondenza di un vettore di punti. Si utilizzi lo script `spoly_eval.m` che richiama tale function e valuta il seguente polinomio

$$p(x) = x^3 - 39x^2 + 504x - 2158 \quad x \in [10, 16].$$

sia in precisione single che double. Considerando il risultato ottenuto in precisione double come esatto, si calcoli e rappresenti graficamente l'errore relativo algoritmico.

Si analizzi il risultato e si individuiono in corrispondenza di quali punti si hanno i valori di maggior errore algoritmico; si dia una spiegazione a quanto trovato.

(**Sugg.** si valuti il polinomio nell'intervallo indicato in punti che siano numeri finiti; questo, per il fatto che i coefficienti del polinomio sono interi, farà sì che gli eventuali errori saranno di tipo algoritmico).

**E. Valutazione polinomio e derivata prima con Ruffini**
Completare la function `poly_eval_der` che implementa l'algoritmo di Ruffini per valutare un polinomio e la sua derivata prima in corrispondenza di un vettore di punti. Si realizzi poi uno script `spoly_eval_der2.m` simile a quello dell'esercizio **C.** per rappresentare gli stessi polinomi graficamente insieme alle loro derivate prime, e che richiami la function `poly_eval_der`.

**F. Metodo di Horner (Esempio 2.3 della dispensa)**
Lo script `test_horner.m` utilizza il metodo di Horner per un polinomio test su un intervallo test. Si tratta di un polinomio a coefficienti rappresentabili esattamente in precisione single e si cerca di valutarlo in punti $x$ che siano numeri finiti; così facendo il problema della valutazione non sarà affetto da errore inerente e gli eventuali errori saranno da imputare all'algoritmo. Si verifichi, utilizzando gli script `conv_bin2dec.m` e `conv_dec2bin.m`, che gli estremi ed i punti di valutazione siano numeri finiti.
Nello script, scommentando/commentando si può abilitare la valutazione in corrispondenza di differenti vettori di numeri finiti.

**G. Sviluppo di Taylor**
La function `taylor_sin` implementa lo sviluppo polinomiale di Taylor di grado $n$ della funzione `sin(x)` centrato in un punto $x_0$. Analizzare lo script per capire cosa è implementato, quindi eseguirlo più volte per differenti punti $x_0$ e gradi $n$ al fine di comprendere il comportamento dell'approssimazione di Taylor grazie alla rappresentazione grafica delle funzioni.

**H. Basi polinomiali e loro rappresentazione grafica**
La function `base_plot.m` implementa la visualizzazione grafica di differenti basi polinomiali definite su un intervallo $[a, b]$. In particolare permette la rappresentazione della base canonica, della base di Bernstein e della base con centro

$$\{1, (x - c), (x - c)^2, \dots (x - c)^n\}$$

con $c$ un punto dell'intervallo di definizione. Dopo aver visionato il codice e le function ivi richiamate, si modifichi la function `base_plot.m` per visualizzare una base alla volta e di ogni base una funzione alla volta.
Si modifichi il codice della base con centro in modo che il centro $c$ sia a scelta l'estremo sinistro $a$, l'estremo destro $b$ o il punto medio $(a + b)/2$ dell'intervallo.

**I. Approssimazione di Weierstrass e Bernstein**
Si realizzi uno script `sapprox_unif.m` che considerata una funzione continua $y = f(x)$ per $x \in [a, b]$, la approssimi con un polinomio nella base di Bernstein di grado $n$ con coefficienti i valori

$$c_i = f(\xi_i) \quad i = 0, \dots, n$$

dove

$$\xi_i = a + \frac{i}{n}(b - a).$$

Si sperimenti questo tipo di approssimazione sulle funzioni degli esercizi A. e B. e per $n \to \infty$. Si rappresenti graficamente sia la funzione che il polinomio approssimante e in una seconda figure la funzione errore assoluto. Cosa si osserva all'aumentare di $n$.

**L. Esercizio di verifica (su Algoritmo di de Casteljau)**
Si completi la function `decast.m` per implementare l'algoritmo di de Casteljau per valutare un polinomio nella base di Bernstein. (Sugg. ci si riferisca al codice Matlab presente sulla dispensa del corso). Realizzare poi uno script `sdecast.m` per valutare e rappresentare graficamente i polinomi nella base di Bernstein i cui coefficienti sono definiti nella function `def_pol.m`.