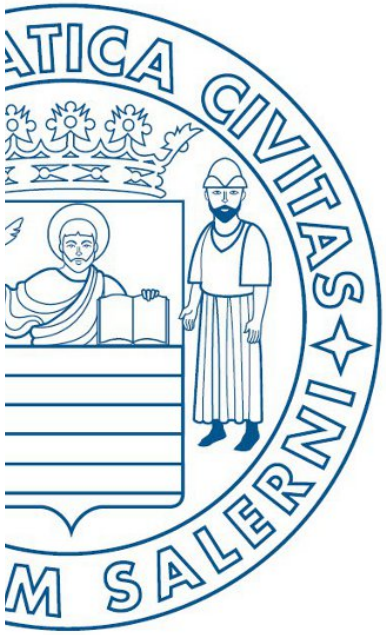


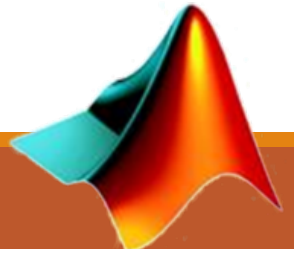
Università di Salerno
Dipartimento di
Ingegneria Industriale

UNIVERSITÀ DEGLI STUDI DI SALERNO

Fondamenti di Informatica

Simulazione Seconda Prova Intercorso - Soluzioni

Prof. Christian Esposito
Corso di Laurea in Ingegneria Meccanica e Gestionale (Classe I)
A.A. 2016/17



MATLAB

Prima Simulazione

Prima Parte – 1/5

- Il candidato consideri i dati organizzati in una matrice **M** ed un array **P**:
  - Una cella di **M** rappresenta semanticamente la quantità di un prodotto (colonna), in una determinata sede del magazzino (riga);
  - Una cella di **P** rappresenta l'importo necessario per uno specifico prodotto.
- La matrice **M** e l'array **P** contengono esclusivamente dati numerici (evidenziati nell'esempio). La matrice e l'array devono essere importati da due file con un apposito script.

M

<<magazzini.txt>>	Pantalone (ind. 1)	Camicia (ind. 2)	Maglia (ind. 3)
Roma (indice 1)	**4**	**25**	**18**
Milano (indice 2)	**15**	**3**	**19**
Napoli (indice 3)	**20**	**10**	**21**

P

<<prezzi.txt>>	Pantalone	Camicia	Maglia
Prezzo	**40**	**30**	**20**

Prima Parte – 2/5

Esercizio 1

- Scrivere una funzione `scorte` che prenda in input la matrice **M** (*magazzini*) e restituisca un array **S**, in cui ogni elemento **S(i)** è definito come segue:

$$S(i) = \begin{cases} 1 & se\ \text{è}\ necessario\ fare\ un\ rifornimento\ nella\ sede\ con\ indice\ i \\ 0 & altrimenti \end{cases}$$

- Si noti che **un rifornimento è necessario** se vi sono uno o più prodotti **la cui quantità** presente in magazzino **è inferiore a 5**.
- Nell'esempio la funzione restituirà $S = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}$
- (**Nota**: E' indifferente che **S** sia un vettore riga o un vettore colonna)

Prima Parte – 2/5

Esercizio 1 – Possibile Soluzione

```
function [S] = scorte(M)
    [nr, nc] = size(M);

    for i=1:nr
        min_sede = min(M(:,i));

        if (min_sede < 5)
            S(i) = 1;
        else
            S(i) = 0;
        end
    end
    S = S';
end
```

Esempio d'uso

```
>> S = scorte(M)

S =

     1
     1
     0
```

Prima Parte – 3/5

Esercizio 2

- Scrivere una funzione chiamata `importi` che prenda in input la matrice **M** (*magazzini*) e l'array **P** (*prezzi*) e restituisca un array **I.**
- Ogni elemento dell'array **I**, contiene l'importo totale contenuto nel magazzino con lo stesso indice.

- **Esempio**
  - I(1) = (4 * 40) + (25 * 30) + (18 * 20) → magazzino Roma
  - I(2) = (15 * 40) + (3 * 30) + (19 * 20) → magazzino Milano
  - I(3) = (20 * 40) + (10 * 30) + (21 * 20) → magazzino Napoli

- (**Nota**: E' indifferente che **I** sia un vettore riga o un vettore colonna)

Prima Parte – 3/5

Esercizio 2 – Possibile Soluzione

```
function [I] = importi(M, P)
    [nr, nc] = size(M);

    for i = 1:nr
        I(i) = sum(M(i,:) .* P);
    end

    I = I';
end
```

Esempio d'uso

```
>> I = importi(M, P)

I =

        1270
        1070
        1520
```

Prima Parte – 4/5

Esercizio 3

- Scrivere una funzione chiamata `piu_fornito` che prenda in input la matrice **M** (*magazzini*) e restituisca l'**indice** del magazzino che ha più prodotti.

Esercizio 4

- Scrivere una funzione chiamata `totale_camicie` che prenda in input la matrice **M** (*magazzini*) e restituisca il **numero totale** di camicie (in tutti i magazzini).

Esercizio 5

- Scrivere una funzione chiamata `minor_incasso` che prenda in input la matrice **M** (*magazzini*) e l'array **P** (*prezzi*) e restituisca l'**indice** del magazzino con il minore dell'importo totale.

Prima Parte – 4/5

Esercizio 3 – Possibile Soluzione

```
function [indice_max] = piu_fornito(M)
    [nr, nc] = size(M);

    max = sum(M(1,:));
    indice_max = 1;

    for i=2:nr
        if max < sum(M(i,:))
            max = sum(M(i,:));
            indice_max = i;
        end
    end
end
```

Prima Parte – 4/5

Esercizio 4 – Possibile Soluzione

```
function [totale] = totale_camicie(M)
    totale = sum(sum(M));
end
```

Esercizio 5 – Possibile Soluzione

```
function [indice] = minor_incasso(M, P)
    [m,n] = size(P);
    if m == 1
        incasso_magazzino = M * P';
    else
         incasso_magazzino = M * P;
    end
    [M, i] = min(incasso_magazzino);
end
```

Prima Parte – 5/5

Esercizio 6

- Scrivere una funzione chiamata `meno_fornito` che prenda in input la matrice **M** (*magazzini*) e l'array **P** (*prezzi*) e restituisca l'**indice** del magazzino e l'**indice** del prodotto tale che in quel magazzino l'importo di quel prodotto è minore rispetto agli altri prodotti nello stesso magazzino e anche negli altri magazzini per lo stesso prodotto o anche prodotti differenti.

Prima Parte – 5/5

Esercizio 6 – Possibile Soluzione

```
function [i,j] = minor_incasso(M, P)
     [m,n] = size(P);
     [mM,nM] = size(M);
     I = zeros(mM,nM);
     if m == 1
          for i = 1:n
               I(:,i) = M(:,i) .* P';
          end
     else
          for i = 1:m
               I(:,i) = M(:,i) .* P;
          end
     end
     [minValues, minIndex] = min(I);
     [minMinValues, index] = min(minValues);
     j = index;
     i = minIndex(index);
end
```

Seconda Parte – 1/2

Esercizio 1

- Sono assegnate le seguenti coppie di valori (x_i;y_i), per i= 1;…;4:

x_i	1	2	3	4
y_i	$\sqrt[3]{2^{22}}$	1	1	$\sqrt[3]{2^{32}}$

- Determinare i parametri α_1 e α_2 in modo che la funzione f(x) = $\alpha_1$10^(α $_2$x) approssimi nel senso dei minimi quadrati i dati (x_i;y_i).

Esercizio 2

- Risolvere il seguente sistema di equazioni lineari, indicando il numero di soluzioni e la loro specifica:

$$\begin{cases} x - y + z = 6 \\ 2x + y - z = -3 \\ x - y - z = 0 \end{cases}$$

Seconda Parte – 1/2

Esercizio 1 – Possibile Soluzione

```
function [alpha] = esercizio_1(x, y)
    p = polyfit(x, log10(y),1)
    alpha = [p(1) 10^(p(2))];
end
```

Esempio d'uso

```
>> x = [1, 2, 3, ;
>> y = [2^(22/3), 1, 1, 2^(32/3)];
>> alpha = esercizio_1(x, y);
```

Esercizio 2 – Possibile Soluzione

```
function [sol] = esercizio_2(A, b)
    rangoA = rank(A);
    rangoAb = rank([A b])
    if rangoA == rangoAb
        disp('Sistema ammette soluzioni')
        if rangoA == length(b)
            disp('Sistema ammette una soluzione')
        else
            disp('Sistema ammette molte soluzioni')
        end
        sol = A\b;
    else
        disp('Sistema non ammette soluzioni')
        sol = NaN;
    end
end
```

Esempio d'uso

```
>> A = [1 -1 1; 2 1 -1;
 1 -1 -1];
>> b = [6; -3; 0];
>> sol = esercizio_2(A,
b)
```

Seconda Parte – 2/2

Esercizio 3

- Calcolare la derivata della seguente funzione nel punto x = 5:

$$f(x) = \frac{e^x + x}{e^x - 5}$$

Seconda Parte – 2/2

Esercizio 3 – Possibile Soluzione

```
function [risultato] = derivata_in_x(fun, punto)
    sysm x;
    fun_der = diff(fun, x);
    risultato = vpa(subs(fun_der, punto), 3);
end
```

Esempio d'uso

```
>> fun = @(x)(exp(x)+x)/(exp(x)-5)
>> punto = 5;
>> risultato = derivata_in_x(fun, punto);
```

Seconda Simulazione

Prima Parte – 1/3

- Il candidato consideri i dati organizzati una matrice **L** ed un array **G**:
  - Una cella di **L** rappresenta semanticamente il numero di libri venduti rivolti ad un certo target (colonna), di un determinata genere (riga);
  - Una cella di **G** rappresenta il guadagno medio per un libro rivolto ad un certo pubblico (colonna)
- La matrice **L** e l'array **G** contengono esclusivamente dati numerici (evidenziati nell'esempio). La matrice e l'array devono essere importati da due file con un apposito script.

L

<<libreria.txt>>	Target 1 (ind. 1)	Target 2 (ind. 2)	Target 3 (ind. 3)
Genere 1 (indice 1)	**15**	**10**	**21**
Genere 2 (indice 2)	**10**	**25**	**4**
Genere 3 (indice 3)	**5**	**21**	**7**

G

<<guadagni.txt>>	Target 1	Target 2	Target 3
Guadagno medio	**2.20**	**3.50**	**2.50**

Prima Parte – 2/3

Esercizio 1

- Scrivere una funzione `libri_venduti` che prenda in input la matrice **L** (*libreria*) ed un intero **i** e restituisca il numero totale di libri venduti per il Target con indice **i**.

Esercizio 2

- Scrivere una funzione `guadagno_medio_libro` che prenda in input la matrice **L** (*libreria*) e l'array **G** (*guadagni medi*) e calcoli e restituisca il guadagno medio per ogni libro venduto.

Esercizio 3

- Scrivere una funzione `genere_meno_venduto` che prenda in input la matrice **L** (*libreria*) e restituisca l'**indice** del genere meno venduto.

Prima Parte – 2/3

Esercizio 1 – Possibile Soluzione

```
function [num_libri_venduti] = libri_venduti(L, i)
    num_libri_venduti = sum(L(:,i));
end
```

Esercizio 2 – Possibile Soluzione

```
function [guadagno_medio] = guadagno_medio_libro(L, G)
    [nr, nc] = size(L);
    somma_libri_venduti = 0;
    somma_guadagno = 0;

    for i = 1:nr
        somma_libri_venduti = somma_libri_venduti + libri_venduti(L, i);
        somma_guadagno = somma_guadagno + (libri_venduti(L, i) * G(i));
    end

    guadagno_medio = somma_guadagno / somma_libri_venduti;
end
```

Prima Parte – 2/3

Esercizio 3 – Possibile Soluzione

```
function [indice] = genere_meno_venduto(L)
    [nr, nc] = size(L);

    min = sum(L(1,:));
    indice = 1;

    for i = 2:nr
        if min > sum(L(i,:))
            min = sum(L(i,:));
            indice = i;
        end
    end
end
```

Prima Parte – 3/3

Esercizio 4

- Scrivere una funzione `target_redditizio` che prenda in input la matrice **L** (*libreria*) e l'array **G** (*guadagni medi*) e restituisca l'**indice** del target più redditizio.

Esercizio 5

- Scrivere una funzione `guadagno` che prenda in input la matrice **L** (*libreria*) e l'array **G** (*guadagni medi*) e restituisca il **valore** complessivo del guadagno per la libreria.

Esercizio 6

- Scrivere una funzione `genere_redditizio` che prenda in input la matrice **L** (*libreria*) e l'array **G** (*guadagni medi*) e restituisca l'**indice** del genere più redditizio.

Prima Parte – 3/3

Esercizio 4 – Possibile Soluzione

```
function [indice] = target_redditizio(L, G)
    [nr, nc] = size(L);
    max = sum(L(:,1)) * G(1);
    indice = 1;
    for i = 2:nr
        if max < sum(L(:,i)) * G(i)
            max = sum(L(:,i)) * G(i);
            indice = i;
        end
    end
end
```

Esercizio 5 – Possibile Soluzione

```
function [risultato] = guadagno(L, G)
    risultato = sum(L*G');
end
```

Prima Parte – 3/3

Esercizio 6 – Possibile Soluzione

```
function [indice] = genere_redditizio(L, G)
    [m, n] = size(G);
    if m == 1
        guadagno_genere = L * G;
    else
        guadagno_genere = L * G';
    end
    [val, index] = min(guadagno_genere);
    indice = index;
end
```

Seconda Parte – 1/2

Esercizio 1

- Supponiamo di aver effettuato le seguenti misurazioni

x	0	1	2	3	4	5	6	7	8	9	10
y	1	1,35	1, 52	1,87	2,0	2,7	3,0	3,7	4,0	4,95	5,2

- Determinare la funzione di approssimazione dei punti assegnati.

Esercizio 2

- Risolvere il seguente sistema di equazioni lineari, indicando il numero delle soluzioni e la loro specifica:

$$\begin{cases} 2x + \frac{1}{2}y = 0 \\ x - y = 1 \\ 3x + 2y = -1 \end{cases}$$

Seconda Parte – 1/2

Esercizio 1 – Possibile Soluzione

```
function [indice] = genere_redditizio(L, G)
    subplot(2,2,1);     plot(x, y);
    subplot(2,2,2);     loglog(x, y)
    subplot(2,2,3);     semilogx(x, y)
    subplot(2,2,4);     semilogy(x, y)
    disp('La quarta ha un andamento lineare');
    coef=polyfit(x,log10(y),1);
    m=coef(1);        b=10^coef(2);
end
```

Seconda Parte – 1/2

Esercizio 2 – Possibile Soluzione

```
function [sol] = esercizio_2(A, b)
    rangoA = rank(A);
    rangoAb = rank([A b])
    if rangoA == rangoAb
        disp('Sistema ammette soluzioni')
        if rangoA == length(b)
            disp('Sistema ammette una soluzione')
        else
            disp('Sistema ammette molte soluzioni')
        end
        sol = A\b;
    else
        disp('Sistema non ammette soluzioni')
        sol = NaN;
    end
end
```

Seconda Parte – 2/2

Esercizio 3

- Data la seguente funzione:

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

- Calcolare il limite per x che tende a +∞ e 0

Seconda Parte – 2/2

Esercizio 3 – Possibile Soluzione

```
function [sol] = esercizio_3(fun, punti)
    sol = ones(1, 2);
    syms x;
    sol(1) = limit(fun, x, punti(1));
    sol(2) = limit(fun, x, punti(2));
end
```

Esempio d'uso

```
>> sysm x;
>> fun = (exp(x)-1)/x
>> punti = [inf 0];
>> risultato = esercizio_3(fun, punti);
```