Politecnico di Milano
Informatica B, AA 2021/2022
Laboratorio 5

Luca Frittoli (`luca.frittoli@polimi.it`)
Mirko Salaris (`mirko.salaris@polimi.it`)

15 Dicembre 2021

1. **Ordinamento negli array** Si scriva una funzione che prenda in input un array `x` e un numero intero `n`, e che restituisca l'`n`-esimo valore più alto di `x`. la funzione dovrà stampare un messaggio d'errore qualora `n` fosse più grande della lunghezza di `x`.

   Soluzione

   ```
   function y = n_esimo(x, n)
       y = nan;
       if ~ isvector(x)
           error('Errore: x deve essere un vettore');
       elseif n <= 0 || ~ isnumeric(n) || floor(n) ~= ceil(n)
           error('Errore: n deve essere un numero intero positivo');
       elseif n > length(x)
           error(['Errore: x ha meno di' num2str(n) ' elementi!']);
       else
           v = sort(x, 'descend');
           y = v(n);
       end
   end
   ```

2. **Permutazioni** Si scriva una funzione `permutazione(v)` che, dato un vettore `v` (riga oppure colonna), controlli se `v` è una permutazione del vettore `[1, 2, 3,..., n-1, n]`, dove `n` è la lunghezza di `v`. In tal caso la funzione deve ritornare `true`, altrimenti `false`. Si ricorda che una permutazione di un vettore è un qualunque riordinamento dei suoi elementi.

   Suggerimento: in altri termini, la funzione deve controllare se 1) `v` contiene tutti gli interi positivi fino a `n` e 2) non ci sono elementi ripetuti.

   Suggerimento 2: può essere utile ordinare l'array prima di fare qualunque controllo.

   Soluzione

   ```
   function result = permutazione(v)
       result = false;
       if ~ isvector(v)
   ```

```
            error('Input errato');
        else
            n = length(v);
            v = sort(v);
            result = isequal(v, 1:n);
        end
    end
```

3. **Tema d'Esame del 25/01/2021 (10 punti)** In tempi di pandemia, il mercato delle consegne a domicilio è molto appetibile. L'azienda Deli-Very vi ha incaricato di produrre alcune componenti di un programma per l'allocazione dei fattorini a ciascuna consegna.

   Gli algoritmi da sviluppare devono considerare le seguenti strutture dati, dove `nFatt` e `nRichieste` rappresentano rispettivamente un numero generico di fattorini e di richieste, rispettivamente: *i)* una matrice `F`, di dimensioni `nFatt`$\times 2$, che contiene le coordinate x e y delle posizioni di ciascun fattorino: l'elemento `F(i,1)` corrisponde alla coordinata x della posizione del fattorino `i`; similmente `F(i,2)` corrisponde alla coordinata y della posizione del fattorino `i`; *ii)* una matrice `M`, di dimensioni `nRichieste`$\times 2$, dove l'elemento `M(i,1)` corrisponde alla coordinata x della posizione del commerciante coinvolto nella richiesta numero `i` e l'elemento `M(i,2)` corrisponde alla coordinata y della posizione di tale commerciante; *iii)* una matrice `C`, di dimensioni `nRichieste`$\times 2$ come la precedente matrice `M`, ma contenente le posizioni x e y del cliente coinvolto in ciascuna richiesta. Come si può notare, le tre matrici hanno la stessa struttura, ma la prima può avere un numero di righe diverso rispetto alle altre due.

   1. Scrivere in linguaggio Matlab una funzione `distmat` che, date due matrici in ingresso come qualsiasi delle precedenti, restituisce una matrice che contiene le distanze euclidee tra tutti i soggetti le cui coordinate sono rappresentate nella prima matrice e quelli le cui coordinate sono rappresentate nella seconda matrice. Ad esempio, `distmat(F, M)` restituisce una matrice `D` dove `D(i,j)` è la distanza fra il fattorino `i` (di cui `F(i, 1)` e `F(i, 2)` rappresentano le coordinate) e il commerciante della richiesta `j` (le cui coordinate sono indicate da `M(j, 1)` e `M(j, 2)`, rispettivamente). La distanza euclidea tra due punti in (x_1, y_1) e (x_2, y_2) è definita nel seguente modo $d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$.
   2. L'azienda decide di accettare solo le richieste di consegna che rispettano la seguente condizione: una richiesta viene accettata se esiste un fattorino la cui distanza dal commerciante coinvolto nella richiesta sia inferiore a quella tra il commerciante e il cliente che ha effettuato la richiesta. Se vi è più di un fattorino, la strategia seleziona quello a distanza minima.

      Si prosegua con la scrittura dello script Matlab del punto precedente in modo da sviluppare i seguenti passi:

      - Si acquisisca da tastiera un numero che indica una richiesta e si assegni questo numero alla variabile `r`.
      - Utilizzando la funzione `distmat` definita nella risposta alla domanda 1, si calcoli la matrice delle distanze tra i clienti in matrice `C` e i commercianti in matrice `M` e si assegni il risultato della funzione alla variabile `J`. Si noti che `J(r, r)` rappresenta la distanza tra il cliente che ha emesso la richiesta `r` e il commerciante che la soddisfa.
      - Sempre utilizzando la stessa funzione, si calcoli la matrice delle distanze tra i fattorini in matrice `F` e i commercianti in matrice `M` e si assegni il risultato della funzione alla variabile `K`. Si noti che `K(:, r)` rappresenta la distanza tra ciascun fattorino e il commerciante che soddisfa la richiesta `r`.
      - Si verifichi se esiste almeno un fattorino in grado di soddisfare la condizione indicata (la sua distanza dal commerciante che soddisfa una richiesta è inferiore rispetto alla distanza tra lo stesso commerciante e il cliente che ha emesso la richiesta). Nel caso in cui non esista, si stampi un messaggio che indichi l'impossibilità per l'azienda di soddisfare la richiesta. Nel caso esista

almeno un fattorino che soddisfa la condizione indicata, si selezioni il fattorino la cui distanza dal commerciante è minima e si stampi a video un messaggio con il numero della richiesta e il numero del fattorino selezionato.

Si scriva la porzione di codice Matlab senza usare cicli, ma utilizzando i confronti e gli operatori logici fra vettori.

Soluzione

```
% parte 1
function D = distmat(A, B)
    M = size(A,1);
    N = size(B,1);
    D = zeros(M,N);
    for ii=1:2
        D = D + (ones(1,N) .* A(:,ii) - B(:,ii)' .* ones(M,1)) .^2;
    end
    D = sqrt(D);
end

% parte 2
% inizializzare a piacere
nFatt = 10;
nRichieste = 7;
F = rand(nFatt, 2);
M = rand(nRichieste, 2);
C = rand(nRichieste, 2);

r = input('Inserire il numero della richiesta ');
J = distmat(C,M);
K = distmat(F,M);
% fattorini che soddisfano la condizione
Fatt = K(:,r) < J(r,r);
% contiamo il numero di fattorini che soddisfano la condizione
if sum(Fatt) == 0
    disp('Impossibile soddisfare la richiesta')
else
    [~, Scelto] = min(K(:,r));
    disp(['Richiesta ' num2str(r) ' assegnata al fattorino ' num2str(Scelto)])
end
```

4. **Matrici ordinate** Si scriva una funzione `righeOrdinate` che prenda in ingresso una matrice e un parametro `direzione` e:

   - con `direzione==0`: restituisce 1 se le righe sono ordinate in modo decrescente, 0 in caso contrario
   - con `direzione==1`: restituisce 1 se le righe sono ordinate in modo crescente, 0 in caso contrario

   Nota che il vettore riga A si dice minore di del vettore riga B se A è minore di B elemento per elemento.

   Esempio:
   Il vettore riga `[1 4 2 8]` è minore di `[2 5 3 9]` ma NON si può dire minore di `[2 5 3 6]`.
   Il vettore riga `[7 2 0 3]` è maggiore di `[0 0 -5 2]` ma NON si può dire maggiore di `[1 1 1 1]`.
   La matrice `[1 4 7 9; 3 6 11 10]` ha le righe ordinate in modo crescente.

Soluzione

```
function fn = righeOrdinate(matrice, direzione)
    % supponiamola ordinata, sovrascriviamo se troviamo "errori"
    fn = 1;
    [nRighe, ~] = size(matrice);
    for k=1:nRighe-1
        if direzione==0 % decrescente
            if ~ all(matrice(k, :) > matrice(k+1, :))
                fn = 0;
            end
        elseif direzione==1 % crescente
            if ~ all(matrice(k, :) < matrice(k+1, :))
                fn = 0;
            end
        else
            error("Parametro 'direzione' errato");
        end
    end
end
```

5. **Campionato automobilistico** Si crei una struttura dati che contenga il nome di alcuni piloti partecipanti a un campionato, e il loro posizionamento in tre Gran Premi: Montecarlo, Silverstone e Monza. Si scriva poi una funzione che prenda in input tale struttura e calcoli la classifica tra i piloti presenti nella struttura, tenendo presente che il vincitore di un Gran Premio ottiene 10 punti, il secondo classificato 9, etc. mentre dalla undicesima posizione in poi non si ottengono punti. La funzione stampa i nomi dei piloti (in ordine di classifica) con a fianco il totale dei punti, e restituisce un array contenente i nomi dei piloti, sempre in ordine di classifica.

Soluzione

```
    function ordine = classifica(campionato)
        punti = zeros(1,length(campionato.piloti));
        punti = punti + max(0, 11 - campionato.Montecarlo);
        punti = punti + max(0, 11 - campionato.Silverstone);
        punti = punti + max(0, 11 - campionato.Monza);
        [C, I] = sort(punti, 'descend');
        ordine = campionato.piloti(I);
        for ii=1:length(campionato.piloti)
            fprintf('%s %d\n', ordine(ii), C(ii));
        end
    end

    % script per testare la funzione
    campionato = struct();
    campionato.piloti = ["David Coulthard","Rubens Barrichello","Pedro de la Rosa"];
    campionato.Montecarlo = [4 1 11];
    campionato.Silverstone = [6 4 3];
    campionato.Monza = [7 12 8];
```

```
ordine = classifica(campionato);
```

6. **Struttura dati per polinomi** Si crei una struttura che contenga i dati necessari per realizzare il grafico di un polinomio di grado qualsiasi. La struttura dovrà contenere i coefficienti del polinomio e un array di ascisse. Si scriva poi una funzione che prenda in input la struttura e generi il grafico del polinomio dato sulle ascisse contenute nella struttura.

Soluzione

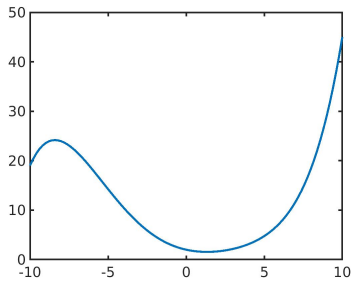
```
function grafico(pol)
    x = pol.dati;
    c = pol.coefficienti;
    y = zeros(size(x));
    for k = 1:length(c)
        y = y + c(k) * x .^ (k - 1);
    end
    plot(x,y);
end

% script per testare la funzione
polinomio = struct();
polinomio.coefficienti = [0, 1, -1, 0.5];
polinomio.dati = linspace(-5, 5);
grafico(polinomio);
```

7. **Plot polinomio** Dopo aver caricato nel vettore `coeff` i dati dal file `coeff.mat`, elimina i coefficienti minori di -100 e quelli maggiori di 100. Poi, facendo uso della funzione `samplePolynomial` riportata sotto, plottarli nell'intervallo $[-10, 10]$. Il risultato deve essere come il grafico in figura.

Download file: Scarica il file al seguente link `https://bit.ly/2021_infoB_coeffmat` e importalo nella cartella attiva su Matlab.

```
function [x , y] = samplePolynomial(polyCoeff, interval)
    a = min(interval);
    b = max(interval);
    x = [a : (b - a) / 100 : b];
    y = zeros(size(x));
    for ii = 1 : 1 : length(polyCoeff)
        y = y + polyCoeff(ii) * x.^(length(polyCoeff) - ii);
    end
end
```



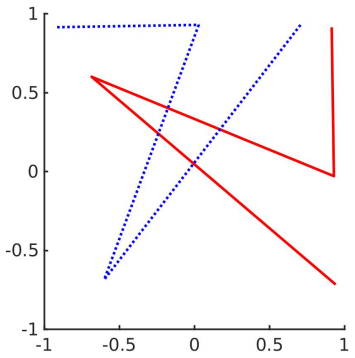
Soluzione

```
load('coeff.mat') % [4e-04, -110, 0, -0.02, 0.3, -0.7, 150, 2]
coeff(coeff < -100 | coeff > 100) = [];
```

```
[x,y]=samplePolynomial(coeff, [-10, 10]);
plot(x,y)
```

8. **Rotazione punti** Scrivi una funzione `ruota` che prende come parametri due vettori di eguale lunghezza che rappresentano le coordinate `x` e le coordinate `y` di punti in uno spazio cartesiano. La funzione ruota i punti intorno all'origine di 90° e restituisce due nuovi vettori, `newx` e `newy`.

Scrivi una funzione `mostraRotazione` che prende come parametri 4 vettori: `x1`, `y1`, `x2`, e `y2`. La funzione genera il grafico mostrando i punti di coordinate `x1,y1` collegati da linee rosse e i punti di coordinate `x2,y2` collegati da linee blu.
Scrivi infine uno script che chiede all'utente quanti punti N inserire, e successivamente per ogni punto chiede le coordinate `x` e `y`. Lo script, utilizzando le funzioni sopra definite, mostra quindi un grafico con le coordinate originali inserite dall'utente e le coordinate ruotate, come in figura.



Nota: la funzione `plot(x, y, 'b')` genera un grafico di colore blu, usando invece `'r'` il grafico sarà di colore rosso. Per maggiori informazioni leggere la documentazione (`help plot` sulla console).

Soluzione

```
%% file ruota.m
function [newx,newy] = ruota(x,y)
    if length(x) == length(y)
        newx = -y;
        newy = x;
    else
        error("Le dimensioni di x e y devono coincidere");
    end
end

%% file mostraRotazione.m
function [] = mostraRotazione(x, y, newx, newy)
    figure;
    hold on;
    plot(x, y, 'r-')
    plot(newx, newy, 'b:')
    hold off;
end

%% script
N = input("Quanti punti vuoi inserire?");
x = zeros(1, N);
y = zeros(1, N);
```

```
for k=1:N
     x(k) = input(sprintf("Coordinata x del punto %d", k));
     y(k) = input(sprintf("Coordinata y del punto %d", k));
end

[newx, newy] = ruota(x, y);
mostraRotazione(x, y, newx, newy);
```

9. **Metodo di bisezione** In alcuni casi è difficile trovare delle soluzioni esatte per equazioni del tipo $f(x) = 0$, dove f è una funzione continua, quindi si ricorre a metodi numerici come il metodo di bisezione. Questo algoritmo sfrutta il Teorema di Bolzano (`https://it.wikipedia.org/wiki/Teorema_di_Bolzano`) per approssimare una soluzione dell'equazione in un intervallo $[a, b]$. Il Teorema afferma che se $f(a)$ e $f(b)$ hanno segno opposto, allora $f(x) = 0$ ha almeno una soluzione nell'intervallo $[a, b]$. Questo fatto può essere sfruttato per localizzare la soluzione, guardando il valore della funzione nel punto medio $m = (a + b)/2$: se, per esempio, $f(a)$ e $f(m)$ hanno segno opposto, allora per il Teorema esiste una soluzione in $[a, m]$, se invece $f(m)$ e $f(b)$ hanno segno opposto, esisterà una soluzione in $[m, b]$. Iterando questo procedimento su $[a, m]$ (o su $[m, b]$, a seconda dei risultati) si arriva ad approssimare una soluzione dell'equazione. Si scriva una funzione che, dati gli estremi di un intervallo $[a, b]$ e una funzione continua f, restituisca – se possibile – una soluzione approssimata dell'equazione $f(x) = 0$ nell'intervallo $[a, b]$. Si controlli infine con un plot la bontà della soluzione trovata.

   Suggerimenti:

   - Per definire in uno script Matlab una funzione, per esempio $f(x) = 2x + x^2$, è possibile utilizzare il comando `f=@(x) 2*x + x.^2`.
   - Per prima cosa è necessario verificare che la funzione f soddisfi le ipotesi del Teorema di Bolzano su $[a, b]$, altrimenti non è possibile applicare il metodo di bisezione.
   - L'algoritmo si ferma tipicamente quando il valore assoluto della funzione nel punto medio dell'intervallo considerato è più piccolo di una certa soglia, ovvero `abs(f(m)) < tol`, dove per esempio `tol = 1e-6`.

Soluzione

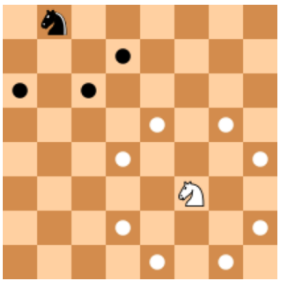
```
function m = bisezione(f, intervallo)
    a = intervallo(1);
    b = intervallo(2);
    if f(a) * f(b) >= 0
        disp("Le ipotesi del teorema non sono soddisfatte");
    else
        cond = true;
        tol = 1e-6;
        while cond
            m = (a+b) / 2;
            cond = abs(f(m)) > tol;
            if f(a) * f(m) < 0
                b = m;
            else
                a = m;
            end
```

```
            end
        end
    end

    % script per testare la funzione
    f = @(x) x.^3 + x + 1;
    intervallo = [-2 2];
    m = bisezione(f, intervallo);
    x = -2:0.1:2;
    y = f(x);
    plot(x,zeros(1,length(x)), '--')
    hold on
    plot(x,y)
    hold on
    plot(m, f(m), 'go')
```

[Nota: il prossimo esercizio potrebbe richiedere un po' di tempo, quindi se non riesci a finirlo nel tempo del laboratorio puoi concluderlo come esercizio per casa.]

10. **Il cavallo negli scacchi** Si scriva una funzione Matlab `min_moves(start,endpos)` che, date in ingresso le coordinate di una casella di partenza `start` e di una casella d'arrivo `endpos` di una scacchiera 8×8, restituisca il numero minimo di mosse con cui un cavallo può raggiungere `endpos` partendo da `start`, se possibile, e restituisca -1 se non è possibile raggiungere `endpos` partendo da `start`. Si utilizzi il seguente script per testare l'implementazione:



```
startlist = [1,1;5,2;1,6;4,4];
outlist = [3,2;6,1;8,8;1,8];
expected = [1,2,5,3];
count = 0;
for ii=1:4
    out = min_moves(startlist(ii,:), outlist(ii,:));
    if out == expected(ii)
        count = count+1;
    end
end
disp(['Passed ', num2str(count), ' tests out of ', num2str(4)]);
```

Suggerimenti:

- negli scacchi, il cavallo si muove a "L", ovvero spostandosi di 2 posizioni in una direzione e 1 nell'altra. Quindi, partendo da una casella della scacchiera, ci sono al massimo 8 caselle raggiungibili dal cavallo in una mossa (vedi figura).
- Può essere utile trattare la scacchiera come una matrice `board` in cui ciascun elemento `board(ii,jj)` è il numero minimo di mosse con cui il cavallo può raggiungere la casella `(ii,jj)` partendo da `start`. Quindi, la casella `start` avrà valore 0, quelle raggiungibili in una mossa da `start` 1 e così via.
- Per decidere se una mossa è valida, è necessario controllare che 1) la casella raggiunta sia all'interno della scacchiera (vedi figura) e 2) la casella raggiunta non sia già stata raggiunta in precedenza (altrimenti non otterremmo il numero minimo di mosse).

Soluzione

```
function out = min_moves(start, endpos)
    board = -ones(8);
    % controlliamo che la casella di partenza sia nella scacchiera
    for i=1:2
       if (start(i) < 1 || start(i) > 8 || endpos(i) < 1 || endpos(i) > 8)
           error('le coordinate devono appartenere alla scacchiera');
       end
    end
    if (sum(floor(start)==ceil(start)) + sum(floor(endpos)==ceil(endpos)) ~= 4)
       error('le coordinate devono essere numeri interi');
    end
    % elenchiamo tutte le possibili mosse del cavallo
    % ogni mossa è un vettore che indica di quante posizioni il cavallo si
    % sposta in orizzontale e in verticale
    moves = [[1,2]; [1,-2]; [-1,2]; [-1,-2]; [2,1]; [2,-1]; [-2,1]; [-2,-1]];
    % impostiamo a 0 il numero di mosse necessarie per arrivare a start
    board(start(1),start(2)) = 0;
    % impostiamo start come primo elemento della sequenza delle posizioni
    % raggiungibili (in questo caso, con zero mosse)
    sequence = start;
    check = false;
    count = 0;
    kk = 1;
    out = -1;
    while (~check && kk ~= 0)
       % aggiorniamo il numero di mosse
       count = count+1;
       kk = 0;
       for j=1:length(sequence(:,1))
           for k=1:length(moves)
               cand = sequence(j,:)+moves(k,:);
               % testiamo la k-esima mossa possibile: deve portare in una
               % casella della scacchiera che non è ancora stata raggiunta
               if (cand(1) >= 1 && cand(1) <= 8 && cand(2) >= 1 && cand(2) <= 8
                    && board(cand(1),cand(2)) == -1)
                   kk = kk+1;
                   board(cand(1), cand(2)) = count;
                   newseq(kk,:) = cand;
                   % controlliamo se la casella raggiunta è endpos
                   if (cand(1) == endpos(1) && cand(2) == endpos(2))
                       check = true;
                   end
               end
           end
       end
       if check
           out = count;
       elseif kk > 0
           sequence = newseq;
```

```
            end
        end
    end
```