Politecnico di Milano
Informatica B, AA 2020/2021
Laboratorio 5

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

3 Dicembre 2020

1. **Numeri di Fibonacci** Si scriva una funzione che prenda in input un intero positivo n e calcoli il corrispondente elemento nella successione di Fibonacci $0, 1, 1, 2, 3, 5, 8, 13, \ldots$.

   Suggerimento: si ricordi che ogni elemento della successione di Fibonacci (eccetto i primi due, che sono per definizione $0, 1$) è la somma dei due precedenti.

   Soluzione

   ```
   function fn = fibonacci(n)
       if n <= 0 || floor(n) ~= ceil(n)
           error('n deve essere un intero positivo');
       elseif n == 1
           fn = 0;
       elseif n == 2
           fn = 1;
       else
           fn = 1;
           f1 = 0;
           for k=3:n
               fn = fn + f1;
               f1 = fn - f1;
           end
       end
   end
   ```

2. **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
```

3. **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
```

4. **Cornici concentriche (dal tema d'esame del 29/01/2018)** Si consideri il seguente problema: si vuole creare una matrice quadrata che sia organizzata nel seguente modo:

```
10  10  10  10  10
10  9   9   9   10
10  9   8   9   10
10  9   9   9   10
10  10  10  10  10
```

   - Si scriva in linguaggio Matlab una funzione iterativa `cornici` che, data la dimensione N della matrice e un numero di partenza P, restituisca al chiamante una matrice quadrata $N \times N$ così definita: la matrice contiene nella cornice più esterna il numero P e numeri decrescenti nelle cornici più interne.
   - Si scriva inoltre uno script in linguaggio Matlab che acquisisca da tastiera la dimensione desiderata N e il numero di partenza P, invochi la funzione `cornici` con gli opportuni parametri e infine stampi a video la matrice risultante.

Soluzione

```
% prima parte
function [M] = cornici(N, P)
    % inizializziamo a P l'intera matrice
    M = P * ones(N);
    ii = 1;
    while ii < N / 2
        % cambiamo i valori nelle cornici più interne
        % (quella più esterna è già inizializzata correttamente)
        M(ii+1 : N-ii, ii+1 : N-ii) = P - ii;
        ii = ii + 1;
    end
end

% seconda parte
N = input('Inserisci la dimensione della matrice: ');
P = input('Inserisci il numero di partenza: ');
M = cornici(N, P)
```

5. **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
```

6. **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);
```

7. **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)
```

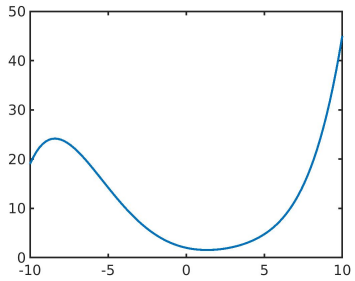
```
            y = y + c(k) * x .^ (k - 1);
        end
        plot(x,y);
    end

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

8. **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/2020_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
```

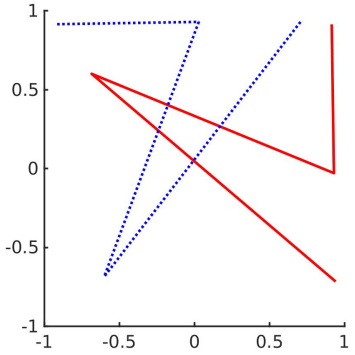


Soluzione

```
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)
```

9. **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);
```

10. **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 poi 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')
```