PROGETTO MATLAB
CALCOLO NUMERICO
A.A. 2016/2017

Michele Di Stefano
0000772273

1.0 CONSEGNA

1.1 Descrizione della consegna

Il progetto consiste nell'implementare in Matlab l'interpolazione spline cubica C^2 nella base di Bernstein con condizioni 'naturale' agli estremi e confrontarla (deve ottenere lo stesso risultato) con la funzione csape di Matlab.
Per le spline cubiche nella base di Bernstein si segua la teoria delle dispense del corso.

Analizzare le possibilità offerte dalla function csape.

Si provi tale interpolante e quanto realizzato per interpolare punti di una funzione ed eventualmente anche punti di curve 2D.

Si preveda la consegna di quanto realizzato (codice, ed esempi) e una piccola relazione sulla function csape e quanto implementato.

1.2 Struttura del progetto

Il progetto è contenuto in proj.tar.gz. La struttura ad albero di directory e file è la seguente:

```
+---proj
|    +--- bs.m
|    +--- errman.m
|    +--- insort.m
|    +--- natcsi.m
|    +--- natcsi2d.m
|    +--- ncs.m
|    +--- test.m
|    +--- README
```

Una volta scaricato l'archivio proj.tar.gz, estrarne il contenuto nella directory corrente eseguendo da riga di comando:

```
tar -zxvf proj.tar.gz
```

1.3 Contenuto dei file

Di seguito una breve descrizione del contenuto dei file. Nelle sezioni successive vedremo più nel dettaglio, sia il codice che l'applicazione di quanto accennato in questo paragrafo.

1.3.1 bs.m

In questo file è presente l'implementazione di una funzione ausiliaria che calcola le funzioni base di Bernstein in un punto, dato un intervallo.

1.3.2 errman.m

E' il file che contiene la gestione degli errori rilevati sull'input.

1.3.3 insort.m

Contiene un algoritmo di ordinamento dei vettori di input.

1.3.4 natcsi.m

Contiene l'implementazione della funzione natcsi, ovvero dell'interpolazione spline cubica.

1.3.5 natcsi2d.m

Contiene l'implementazione della funzione natcsi2d, ovvero la determinazione di curve tramite spline cubica.

1.3.6 ncs.m

Contiene l'implementazione della parte principale del progetto, ovvero l'interpolazione spline cubica con condizioni naturali agli estremi.

1.3.7 test.m

E' un file che contiene diversi test al fine di verificare il funzionamento delle funzioni natcsi e natcsi2d.

1.3.8 README

Contiene informazioni sul progetto, sul suo utilizzo e sull'autore.

2.0 INTERPOLAZIONE SPLINE CUBICA

2.1 Teoria ed implementazione: natcsi

2.1.1 Cubic Spline

2.1.1.1 Parte teorica

Nel seguente paragrafo vedremo di pari passo le basi teoriche dell' interpolazione spline cubica e la rispettiva implementazione in ambiente MATLAB. La teoria è tratta dal materiale didattico fornito (Casciola, 2016).

Per interpolazione globale si intende un metodo che determina il polinomio a tratti $pp(x)$ interpolante ricavando i polinomi $p_i(x)$ in modo non indipendente dagli altri.

Se abbiamo i dati $(x_i, y_i), i = 0, ..., m$, allora una spline cubica $s(x)$ è data dalla funzione

$$s(x) = \begin{cases} p_0(x) & x \in [x_0, x_1] \\ p_1(x) & x \in [x_1, x_2] \\ \vdots & \\ p_{m-1}(x) & x \in [x_{m-1}, x_m] \end{cases}$$

Dove ogni $p_i(x)$ è un polinomio cubico. Scriviamo i $p_i(x)$ nella base di Bernstein nel seguente modo:

$$p_i(x) = q_i B_{0,3}(x) + r_i B_{1,3}(x) + s_i B_{2,3}(x) + t_i B_{3,3}(x)$$

Per determinare la spline dobbiamo determinare i coefficienti q_i,r_i ,s_i et_i per ogni$i$. Poiché ci sono m intervalli ci sono $4m$ coefficienti da determinare. Prima richiediamo che la spline interpoli i dati:

$$p_i(x_i) = y_i$$
$$p_i(x_{i+1}) = y_{i+1}$$

Per ogni $i = 0, ..., m - 1$.

In altre parole dovrà essere:

$$q_i = y_i$$
$$t_i = y_{i+1} \; i = 0, ..., m-1$$

Per cui

$$p_i(x) = y_i B_{0,3}(x) + r_i B_{1,3}(x) + s_i B_{2,3}(x) + y_{i+1} B_{3,3}(x) \quad i = 0, ..., m-1 \quad (2.1)$$

Affinché poi due polinomi consecutivi si raccordino in C^2 devono essere soddisfatte le seguenti ulteriori condizioni

$$p'_{i-1}(x_i) = p'_i(x_i)$$
$$p''_{i-1}(x_i) = p''_i(x_i)$$

su tutti i punti interni, cioè per $i = 1, ..., m-1$, che possiamo esplicitare nel seguente insieme di condizioni

$$\frac{y_i - s_{i-1}}{h_{i-1}} = \frac{r_i - y_i}{h_i}$$
$$\frac{y_i - 2s_{i-1} + r_{i-1}}{h_{i-1}^2} = \frac{s_i - 2r_i + y_i}{h_i^2} \quad (2.2)$$
$$i = 1, , m-1$$

Si tratta di $2m-2$ equazioni lineari nelle $2m$ incognite r_i ed s_i , $i = 0, ..., m-1$. Solitamente si aggiungono due ulteriori condizioni, per esempio

$$p'_0(x_0) = p'_{m-1}(x_m) = 0$$

Queste sono chiamate condizioni **naturali**.

In seguito all'osservazione che le condizioni derivano dalla definizione di $p_i(x) = 0, ..., m-1$ e dalle espressioni delle loro derivate prima e seconda, se poniamo

$$D_i = \frac{y_i - s_{i-1}}{h_{i-1}} = \frac{r_i - y_i}{h_i} \quad (2.3)$$

possiamo riscrivere le seconde condizioni 2.2 solo in termini di D_i e y_i sostituendo al posto di s_i ed r_i le espressioni trovate mediante la 2.3; sarà:

$$\frac{y_{i-1} - y_i + 2D_i h_{i-1} + D_{i-1} h_{i-1}}{h_{i-1}^2} = \frac{y_{i+1} - y_i - D_{i+1} h_i - 2D_i h_i}{h_i^2}$$

moltiplicando entrambi i membri per h_i h_{i-1} e raccogliendo opportunamente i termini si ottiene:

$$h_i D_{i-1} + 2D_i(h_i + h_{i-1} D_{i+1} = \frac{h_i}{h_{i-1}}(y_i - y_{i-1}) + \frac{h_{i-1}}{h_i}(y_{i+1} - y_i)$$

Per $i = 1, ..., m-1$
Abbiamo quindi un sistema $(m-1) \times (m-1)$ che avrà una ed una sola soluzione

$$\begin{pmatrix} 2(h_1 + h_0) & h_0 & 0 & \dots & 0 \\ h_2 & 2(h_2 + h_1) & h_1 & \dots & 0 \\ \dots & & & & \\ 0 & \dots & 0 & h_{m-1} & 2(h_{m-1} + h_{m-2}) \end{pmatrix} \begin{pmatrix} D_1 \\ D_2 \\ \vdots \\ D_{m-1} \end{pmatrix} =$$

$$= \begin{pmatrix} \frac{h_1}{h_0}(y_1 - y_0) + \frac{h_0}{h_1}(y_2 - y_1) - h_1 D_0 \\ \frac{h_2}{h_1}(y_2 - y_1) + \frac{h_1}{h_2}(y_3 - y_2) \\ \vdots \\ \frac{h_{m-1}}{h_{m-2}}(y_{m-1} - y_{m-2}) + \frac{h_{m-2}}{h_{m-1}}(y_m - y_{m-1}) - h_{m-2} D_m \end{pmatrix}.$$

Determinate le $D_1, D_2, , D_{m-1}$ insieme alle assegnate D_0 e D_m si può procedere al calcolo degli m polinomi che costituiscono la polinomiale cubica a tratti di interpolazione. Tali polinomi nella forma (2.1) saranno dati dai coefficienti

$$q_i = y_i, r_i = y_i + h_i D_i, s_i = y_{i+1} - h_i D_{i+1}, t_i + 1 = y_{i+1} \quad i = 0, ..., m-1$$

2.1.1.2 Parte implementativa

2.1.1.2.1 Interfaccia natcsi

La funzione prende in input due vettori: x, il vettore delle osservazioni; y, il vettore dei valori da interpolare. Restituisce al chiamante un vettore pp nel quale sono memorizzati i valori della spline cubica.

```
function  [ pp ]  = natcsi( x, y)
```

2.1.1.2.2 Implementazione

Il vettore delle osservazioni viene passato alla funzione natcsierr, la quale si occupa di verificare che l'input sia corretto, ed eventualmente riporta gli errori.

```
x = errman(x,y);
```

Si verifica che il vettore delle osservazioni sia in ordine non decrescente, eventualmente viene ordinato. In quest'ultimo caso si aggiorna anche il vettore y per mantenere la corrispondenza delle coppie (x_i, y_i).

```
if any(diff(x)<0)
    [x,y] = insort(x,y);
end
```

Si trasferisce il controllo alla funzione ncs, passando i vettori x ed y come argomenti.

```
[pp, f] = ncs(x,y);
```

2.1.1.2.2.1 ncs

In m viene salvato il numero delle osservazioni che corrisponde alla lunghezza del vettore x.

```
m = length(x);
```

Si definiscono gli incrementi $h_i = x_{x+1} - x_i$

```
h = zeros(1, m);
for i = 1:m-1
    h(i) = x(i+1)-x(i);
end
```

Si definisce il sistema lineare a diagonale dominante, partendo dalla matrice A.
Prima vengono impostate le condizioni naturali agli estremi della matrice,

```
A = zeros(m);
A(1, 1:2) = [2*h(1) h(1)];
A(m, m-1:m) = [h(m-1) 2*h(m-1)];
```

successivamente vengono popolati i restanti elementi della matrice.

```
c = 1;
for r = 2:m-1
    A(r,c) = h(r);
    A(r,c+1) = 2*(h(r)+h(r-1));
    A(r,c+2) = h(r-1);
    c = c+1;
end
```

Si procede similmente con il vettore colonna b.
Si definiscono le condizioni agli estremi del vettore colonna,

```
b = zeros(m,1);
b(1,1) = y(2)-y(1);
b(m,1) = y(m)-y(m-1);
```

e si popolano i restanti elementi del vettore colonna.

```
for i = 2:m-1
    b(i,1) = (h(i)/h(i-1))*(y(i)-y(i-1))+((h(i-1)/h(i))*(y(i+1)-y(i)));
end
```

Si è definito un sistema lineare del tipo $Ad = b$ dove d è il vettore colonna delle D_i, incognite necessarie per calcolare i coefficienti dei polinomi di Bernstein.
Si passa alla risoluzione del sistema lineare.

```
d = A\b;
```

Il vettore d è utilizzato per calcolare i coefficienti dei polinomi di Bernstein, memorizzati poi in 4 vettori distinti, uno per ogni collezione di coefficienti: si definiscono quindi i vettori qc, rc, sc, tc che contengono i rispettivi q_i, r_i, s_i, t_{i+1} per $i = 0, m-1$.

```
rc = zeros(1,m-1);
sc = zeros(1,m-1);
for i = 1:m-1
    rc(i) = y(i)+h(i)*d(i);
    sc(i) = y(i+1)-h(i)*d(i+1);
end

rc = zeros(1,m-1);
sc = zeros(1,m-1);
for i = 1:m-1
    rc(i) = y(i)+h(i)*d(i);
    sc(i) = y(i+1)-h(i)*d(i+1);
end
```

Si può ora procedere con la valutazione della spline.
L'intervallo delle osservazioni viene suddiviso in un maggior numero di intervalli.

```
f = linspace(min(x),max(x), 100);
```

Per ogni punto nell'intervallo f creato precedentemente, si controlla a che sottointervallo $[x_i, x_{i+1}]$ appartiene, in modo tale da utilizzare i coefficienti appropriati nella valutazione del polinomio di Bernstein. I valori risultanti dalle valutazioni sono memorizzati nel vettore pp.

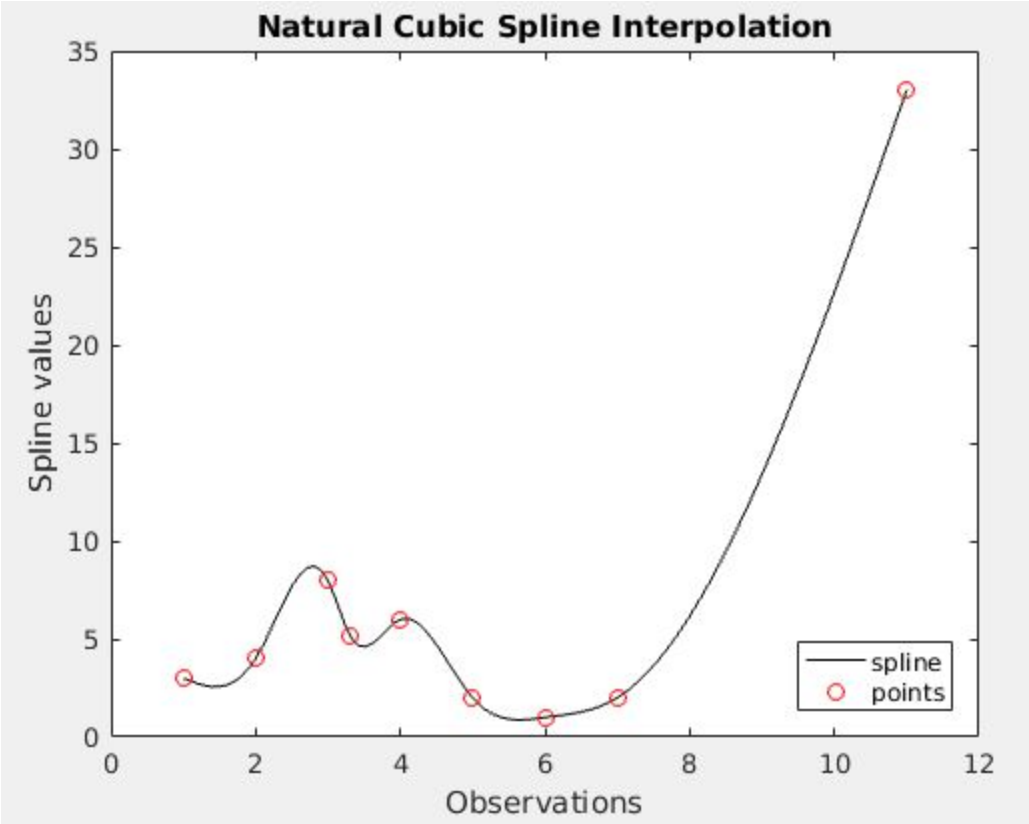
```
pp = double.empty;
for i = 1:m-1
    for j = 1:length(f)
        if(f(j)>=x(i)) && (f(j)<=x(i+1))
            pp(j) = qc(i)*bs(0,3,f(j),x(i),x(i+1)) + rc(i)*bs(1,3,f(j),x(i),x(i+1))
            + sc(i)*bs(2,3,f(j),x(i),x(i+1)) + tc(i)*bs(3,3,f(j),x(i),x(i+1));
        end
    end
end
```

La valutazione del polinomio di Bernstein avviene utilizzando una funzione ausiliaria (bs) che valuta le funzioni di base di Bernstein.

Per visualizzare il grafico è sufficiente plottare f e pp.

```
plot(f,pp,'k',x,y,'ro')
```

2.1.2 Esempi



3.0 CURVE CUBICHE A TRATTI DI INTERPOLAZIONE

3.1 Teoria ed implementazione: natcsi2d

3.1.1 Curve cubiche a tratti di interpolazione

3.1.1.1 Parte teorica

Siano dati $m+1$ punti $Q_i \equiv (x_i, y_i)\ i = 0, ..., m$; si vogliono determinare la curva cubica a tratti C^1 e la spline cubica che interpolano i punti Q_i ; si cerca una curva in forma parametrica

$$\mathbf{C}(t) = \begin{pmatrix} C_1(t) \\ C_2(t) \end{pmatrix}$$

tale che siano verificate le condizioni di interpolazione:

$$\begin{cases} C_1(t_i) = x_i \\ C_2(t_i) = y_i \end{cases} \quad i = 0, \ldots, m$$

Il problema di interpolazione con una curva (funzione vettoriale a due componenti) si riduce a due problemi di interpolazione con funzioni scalari, cioè si cercano: la funzione $C_1(t)$ che interpola i punti (t_i, x_i) e la funzione $C_2(t)$ che interpola i punti (t_i, y_i). Si noti che i valori parametrici t_i in corrispondenza dei quali interpolare non sono assegnati e devono essere determinati come parte del problema stesso.

Un metodo è quello della parametrizzazione della corda. Questo metodo tiene conto della geometria dei punti, poiché si mantengono invariati i rapporti:

$$\frac{t_{i+1} - t_i}{t_{i+2} - t_{i+1}} = \frac{||Q_{i+1} - Q_i||_2}{||Q_{i+2} - Q_{i+1}||_2}$$

Per determinare i valori t_i si può procedere come segue:

$$\tau_0 = 0$$
$$\tau_i = \tau_{i-1} + \sqrt{(x_i - x_{i-1})^2 + (y_i - y_{i-1})^2} \quad i = 1, ..., m$$

Infine si ottengono i valori:

$$t_i = \frac{\tau_i}{\tau_m} \ i = 1, ..., m$$

Determinati i parametri t_ii , i metodi più utilizzati sono l'interpolazione polinomiale cubica a tratti C^1 e spline cubica che fornisce una curva C^2. E' stato utilizzato il secondo metodo.

3.1.1.2 Parte implementativa

3.1.1.2.1 Interfaccia natcsi2d

La funzione prende in input due vettori: x,y , che rappresentano i punti $Q_i(x_i, y_i) \ i = 0, ..., m$. Restituisce al chiamante un vettore dd di dimensioni $2 \times length(c1)$ nel quale sono memorizzati i valori delle due funzioni scalari $C_1(t)$ e $C_2(t)$.

```
function  [dd] = natcsi2d( x, y )
```

3.1.1.2.2 Implementazione

Si inizia procedendo come per la natcsi, dunque con il controllo degli errori sull'input.

```
x = errman(x,y);
```

Si calcola il numero di osservazioni passate come input.

```
m = length(x);
```

Si implementa il metodo di parametrizzazione della corda memorizzando i τ_i in un vettore, e calcolando di conseguenza il vettore t_i.

```
t = zeros(1, m);

tau = zeros(1, m);
tau(1,1) = 0;

for i = 2:m
    tau(1, i) = tau(1, i-1) + sqrt((x(i)-(x(i-1)))^2 + (y(i)-y(i-1))^2);
end

for i = 1:m
    t(1,i) = tau(i)/tau(m);
end
```

Si determinano le funzioni scalari: $C_1(t)$ che interpola i punti (t_i, x_i), e $C_2(t_i)$ che interpola i punti (t_i, y_i). Per determinarle si eseguono due chiamate successive alla ncs. In una chiamata si passano come argomenti i vettori (t_i, x_i) e nell'altra si passano i vettori (t_i, y_i). I due vettori $c1$ e $c2$ vengono salvati in un vettore dd che viene restituito al chiamante.

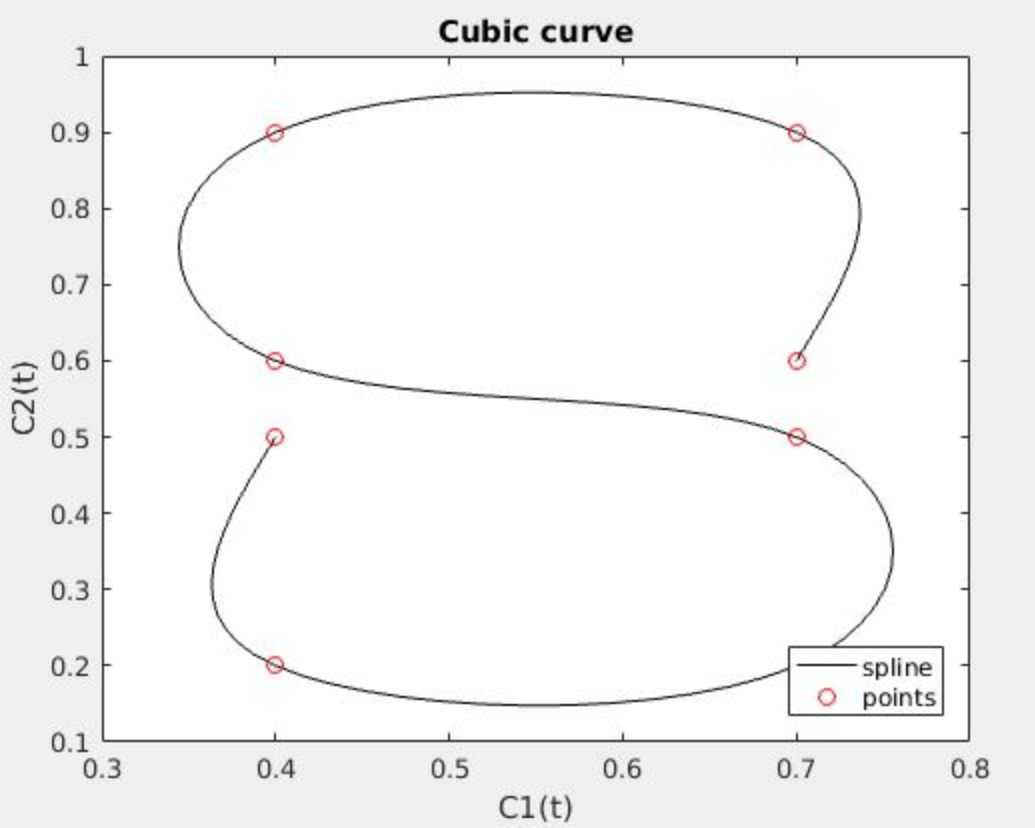
```
c1 = natcsi(t, x);
c2 = natcsi(t, y);

dd = zeros(2, length(c1));
dd(1,:) = c1;
dd(2,:) = c2;
```

Per visualizzare il grafico è sufficiente plottare $c1$ e $c2$.

```
plot(c1,c2,'k',x,y,'or')
```

3.1.2 Esempi


Cubic curve
C2(t)
C1(t)
spline
points
1
0.9
0.8
0.7
0.6
0.5
0.4
0.3
0.2
0.1
0.3
0.4
0.5
0.6
0.7
0.8

4.0 FUNZIONI AUSILIARIE

4.1 bs

4.1.1 Parte teorica

Valuta le funzioni di base di bernstein utilizzando la seguente formula:

$$B_{i,n}(x) = \binom{n}{i} \frac{(b-x)^{n-i}(x-a)^i}{(b-a)^n} \quad i = 0, \ldots, n$$

4.1.2 Parte implementativa

4.1.2.1 Interfaccia bs

Restituisce al chiamante il valore delle funzione di base di Bernstein $B_{i,n}(x)$ con $x \in [a, b]$, memorizzandolo in s.

```
function [s] = bs(i, n, x, a, b)
```

4.1.2.2 Implementazione

Si utilizza una delle funzioni di MATLAB, in particolare la $nchoosek$, che calcola il coefficiente binomiale, per il resto l'implementazione è diretta.

```
s = nchoosek(n,i)*(((b - x)^(n-i)*(x-a)^i)/(b-a)^n);
```

4.2 errman

4.2.1 Parte implementativa

4.2.1.1 Interfaccia errman

La funzione prende in input i due vettori x ed y e, se non rileva errori, restituisce il vettore x con le opportune modifiche, altrimenti produce uno specifico messaggio di errore.

```
function [ newx ] = errman(x,y)
```

4.2.1.2 Implementazione

4.2.1.2.1 XorYempty

Verifica che nessuno dei 2 vettori in input sia vuoto.

```
if(isempty(x) || isempty(y))
    error('NATCSI:XorYempty', 'Error: one or both vectors are empty' )
end
```

4.2.1.2.2 Xnotreal

Verifica che x sia un vettore di numeri reali, eventualmente ignora la parte immaginaria, aggiornando il vettore x.

```
if ~all(isreal(x))
   x = real(x);
   warning('NATCSI:Xnotreal', 'Warning: data points imaginary part will be ignored')
end
```

4.2.1.2.3 Nans

Verifica che non siano presenti valori NaN o infiniti.

```
nanx = find(~isfinite(x), 1);
if ~isempty(nanx)
    error('NATCSI:NaNs', 'Error: there are infinite or NaN values among the sites')
end
```

4.2.1.2.4 XYnotsamenum

Verifica che i vettori x ed y siano della stessa lunghezza.

```
if(length(x) ~= length(y))
    error('NATCSI:XYnotsamenum', 'Error: x and y have a different number of elements')
elseif(length(x) == 1)
    error('NATCSI:toofewpoints', 'Error: you need more than one point to interpolate')
end
```

Infine viene memorizzato il vettore x aggiornato nel vettore $newx$.

```
newx = x;
```

4.3 insort.m

4.3.1 Parte implementativa

4.3.1.1 Interfaccia insort

La funzione insort prende in input i due vettori x ed y e garantisce che le osservazioni siano in ordine non decrescente.

```
function [newx, newy] = insort(x, y)
```

4.3.1.2 Implementazione

Si rileva il numero di osservazioni. Il vettore x viene memorizzato in un secondo vettore $oldx$. Il vettore x viene ordinato in modo crescente tramite la funzione $sort$.
Successivamente viene ripristinata la corrispondenza tra le coppie (x_i, y_i), cercando gli indici delle y corrispondenti alle x presenti in $newx$, nel nuovo ordine.

```
m = length(x);

oldx = x;
newx = sort(x);

for i = 1:m
    for j = 1:m
        if(newx(i) == oldx(j))
            newy(i) = y(j);
        end
    end
end
```

5.0 CSAPE

5.1 Possibilità di utilizzo

5.1.1 Interfaccia

La csape è la funzione di MATLAB che implementa l'interpolazione spline cubica con condizioni agli estremi.
pp è la polynomial piecewise form (forma polinomiale a tratti) della spline cubica, rappresentata, in MATLAB, da una struct.

```
pp = csape(x,y)
pp = csape(x,y,conds)
```

5.1.2 Applicazioni

5.1.2.1 Default (Lagrange)

Nella sua versione più semplice

```
pp = csape(x,y)
```

Utilizza la condizione di Lagrange, alternativa all'interpolante not-a-knot utilizzata dalla csapi.

5.1.2.2 Condizioni Naturali

Possiamo selezionare altre condizioni agli estremi, ad esempio 'variational' (oppure 'second'):

```
pp = csape(x,y,'variational')
```

utilizza le condizioni naturali, ovvero la derivata seconda è zero ai due estremi.

5.1.2.3 Specificare le derivate seconde

Si possono utilizzare esplicitamente le derivate seconde corrette, in modo da avere un errore più piccolo. Ad esempio:

```
pp = csape(x,[endcond(1) subplus(x-2).^3 endcond(2)], 'second');
```

Dove $endcond(1)$ ed $endcond(2)$ utilizzando agli estremi i valori della derivata seconda, di $f(x) = ((x - x_i)_+)^3$, da noi calcolata.

5.1.2.4 Specificare le pendenze

E' possibile specificare le pendenze agli estremi. E' la clamped cubic spline. Ad esempio:

```
pp = csape(x,[sl,y,sr],'clamped')
```

Crea una spline cubica interpolante i dati (x, y) e con una pendenza $s1$ nell'estremo sinistro e con una pendenza sr nell'estremo destro.

5.1.2.5 Condizioni periodiche

E' possibile specificare condizioni periodiche agli estremi utilizzando 'periodic'.

```
pp = csape(x,y, 'periodic' );
```

5.1.2.6 Not a Knot

Rende nodi inattivi il secondo ed il penultimo sito, ignorando eventuali valori di condizioni agli estremi.

5.1.2.7 Tabella riassuntiva

`'complete'` or `'clamped'`	Match endslopes (as given, with default as under "default").
`'not-a-knot'`	Make second and second-last sites inactive knots (ignoring end condition values if given).
`'periodic'`	Match first and second derivatives at left end with those at right end.
`'second'`	Match end second derivatives (as given, with default [0 0], i.e., as in `'variational'`).
`'variational'`	Set end second derivatives equal to zero (ignoring end condition values if given).
default	Match endslopes to the slope of the cubic that matches the first four data at the respective end (i.e., Lagrange).

Esempi: https://it.mathworks.com/help/curvefit/examples/cubic-spline-interpolation.html

6.0 BIBLIOGRAFIA

G. Casciola. *Dispensa di Calcolo Numerico. Metodi numerici per il calcolo*. A.A 2016/2017

https://it.mathworks.com