MchiN strisce

programma di analisi delle sezioni a strisce

Andrea Marchi

28 maggio 2021

QUESTO DOCUMENTO NON É ANCORA COMPLETO

Indice

1 Programma

Il programma consiste in una classe (`MchiNstrisce`) implementata in *MATLAB*. Basta inserire il file corrispondente alla definizione della classe (il file "`MchiNstrisce.m`") nella cartella del file che la chiama, o in una delle sottocartelle nel *path* di MatLab per utilizzare la classe e tutte le sue funzionalità.

1.1 Teoria di base

L'analisi a fibre consiste nel dividere la sezione in "piccole" aree chiamate *fibre*[1]. Le *strisce*, usate in questa classe come discretizzazione delle sezioni, non sono altro che fibre larghe quanto la larghezza della sezione ad una determinata y (secondo l'asse perpendicolare all'asse di sollecitazione)[2].

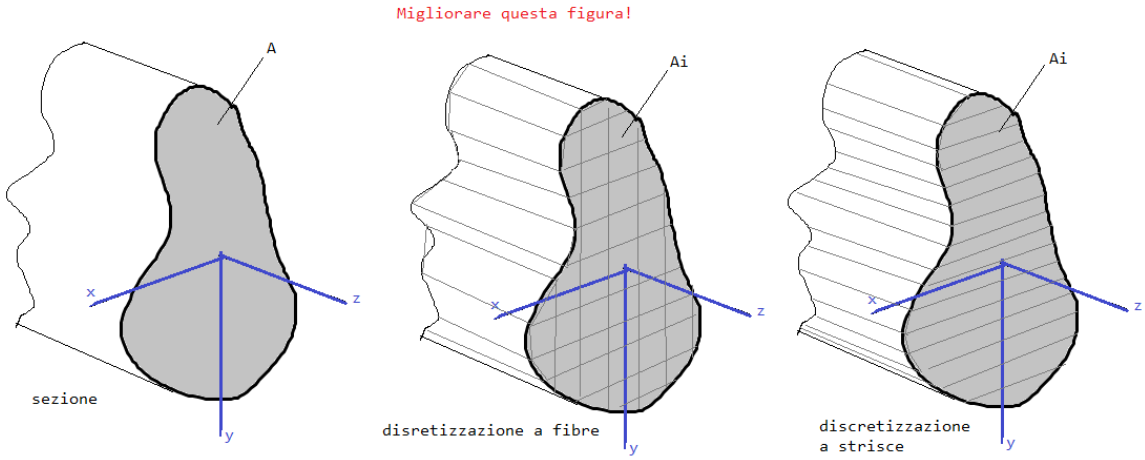


Figura 1: Discretizzazione della sezione in *fibre* e in *strisce*.

in realtà la y l'ho presa verso l'ALTO io (il contrario della convenzione italiana

Le equazioni che definiscono le caratteristiche di sollecitazione interne ad un elemento, a partire dalle tensioni interne al continuo, sono:

$$N_{int}(z) = \int_A \sigma_z(x,y,z)\, dA \qquad V_{int}(z) = \int_A \tau_{zy}(x,y,z)\, dA \qquad M_{x,int}(z) = \int_A \sigma_z(x,y,z)\, y\, dA \quad (1)$$

dove gli integrali sono integrali superficiali svolti sulla sezione della trave (nel piano xy). Dato che in generale la forma della sezione (e quindi il dominio di integrazione) non è esprimibile in forma analitica in funzione di x e y, si ricorre alla discretizzazione della sezione in elementi piccoli (ma non infinitesimi) approssimando gli integrali in somme. Tale discretizzazione permette di calcolare anche sezioni costituite da materiali eterogenei e con legami costitutivi non-lineari (come il calcestruzzo armato).

$$N_{int} = \sum_i \sigma_z(x_i,y_i)\, A_i \qquad V_{int} = \sum_i \tau_{zy}(x_i,y_i)\, A_i \qquad M_{x,int} = \sum_i \sigma_z(x_i,y_i)\, y_i\, A_i \quad (2)$$

dove x_i e y_i sono le coordinate del baricentro della fibra (sempre rispetto al baricentro della sezione omogeneizzata G). Inoltre la coordinata z è stata omessa in quanto si è interessati a valutare la resistenza della sezione, che generalmente non varia al variare della lunghezza dell'elemento.

immagine di una sezione piana discretizzata con le quote di b_i con vari materiali e anche dei buchi per mostrare le varie opportunità

[1] Riferimento al modello di Drukcer

[2] In questo caso l'asse di sollecitazione è l'asse y e quindi la larghezza della fibra i-esima è la larghezza complessiva della sezione (per quel determinato materiale) all'"altezza" y_i della fibra.

L'ipotesi principale utilizzata nel calcolo delle azioni interne è che *le sezioni piane e perpendicolari all'asse dell'elemento rimangono piane e perpendicolari anche nella configurazione deformata*. Dunque l'analisi della sezione a fibre consiste nel determinare le azioni interne (N_{int}, V_{int} e $M_{x,int}$) in funzione della deformazione, tramite i legami costitutivi dei vari materiali. Fare questo calcolo in maniera numerica per ogni striscia permette di utilizzare anche legami costitutivi non-lineari, che risulta particolarmente conveniente per elementi in calcestruzzo armato dove il comportamento non-lineare è predominante. Per trovare la deformazione corrispondente ad un determinato sistema di azioni esterne (N, V, e M_x) si utilizzano le equazioni di equilibrio[3]

$$N_{int} = N \qquad V_{int} = V \qquad M_{x,int} = M_x \tag{4}$$

insieme ad algoritmi di soluzione per problemi non-lineari. Gli algoritmi presi in considerazione in questa trattazione sono l'algoritmo di *bisezione* e quello di *Newton-Raphson*.

1.2 Geometria della sezione

1.3 Situazioni limite

Grafico $M - \chi$ in cui sono mostrati i punti limite: Punto di primo snervamento, snervamento e rottura

1.4 Punto di fessurazione

La fessurazione avviene quando la fibra più allungata supera l'allungamento di rottura del calcestruzzo ε_{ct}. Per calcolare correttamente le azioni interne nel punto di fessurazione bisogna utilizzare un legame costitutivo per il calcestruzzo che includa la descrizione del comportamento a trazione.

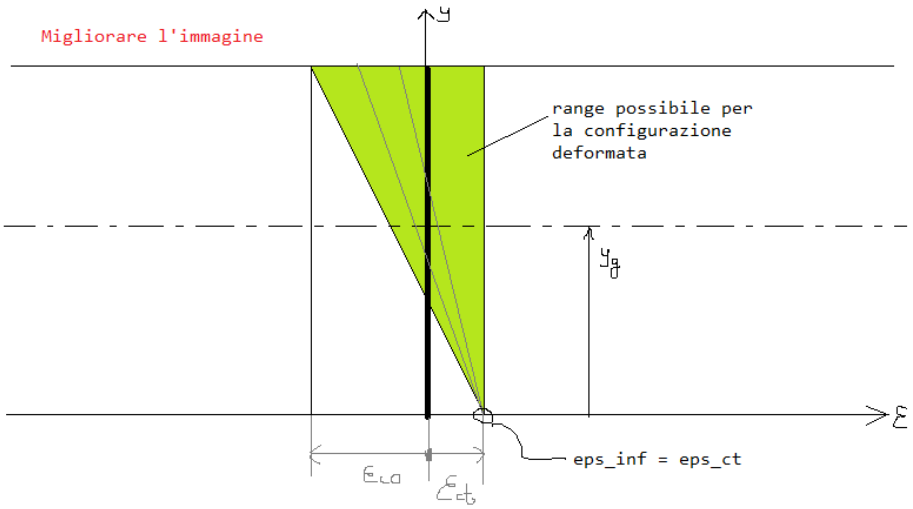


Figura 2: Intervallo possibile per la configurazione deformata nella condizione di prima fessurazione.

[3]Da notare che, in generale, le equazioni di equilibrio impongono che la somma delle forze e dei momenti devono essere uguali a zero

$$\text{equilibrio alla traslazione: } \rightarrow \sum \mathbf{f}_i = \mathbf{0} \qquad \text{equilibrio alla rotazione: } \rightarrow \sum \mathbf{m}_i = \sum \mathbf{f}_i \times \mathbf{d}_i = \mathbf{0} \tag{3}$$

In questo caso tuttavia le azioni interne sono da considerarsi come delle *reazioni*. Dunque la condizione di equilibrio è un'eguaglianza tra azioni (N, V, e M_x) e reazioni (N_{int}, V_{int} e $M_{x,int}$).

1.4.1 Punto di snervamento

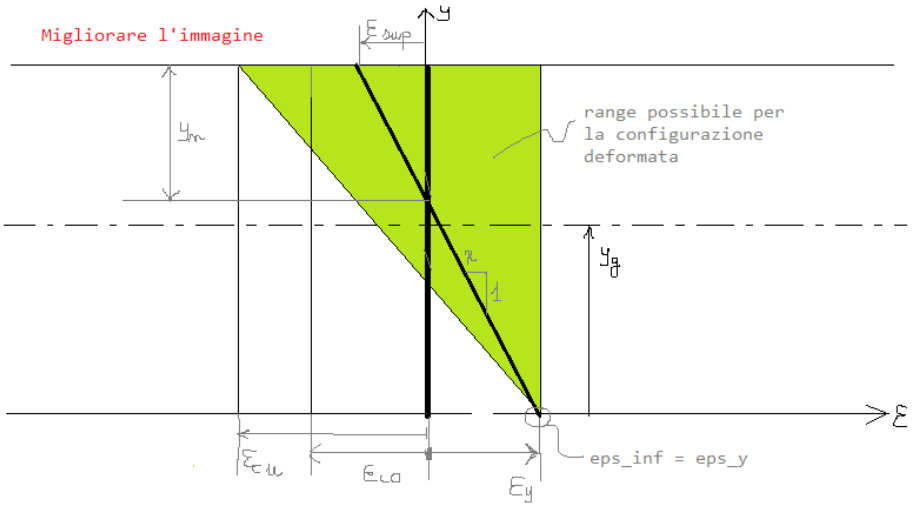


Figura 3: Intervallo possibile per la configurazione deformata nella condizione di snervamento.

1.4.2 Punto di rottura

La *rottura* di una sezione si raggiunge quando il *calcestruzzo (o l'acciaio) raggiunge la sua deformazione ultima*.

Dato che, secondo l'ipotesi cinematica principale, *le sezioni piane rimangono piane* il campo di deformazione della sezione è definito da due parametri. Dunque, definiti dei legami costitutivi per il calcestruzzo e l'acciaio, anche le azioni interne N_{int} e M_{int} dipendono da due parametri. In questo particolare caso risulta conveniente prendere come parametri la deformazione al lembo inferiore ε_{inf} e la deformazione al lembo superiore ε_{sup}. La deformazione, in funzione della profondità y, è definita da

$$\varepsilon(y; \varepsilon_{inf}, \varepsilon_{sup}) = \frac{\varepsilon_{sup} - \varepsilon_{inf}}{H}\, y + \varepsilon_{inf} \tag{5}$$

Immagine con la deformazione nel piano yz e anche la sezione affianco con la fibra messa in evidenza

Dunque lo gli sforzi interni sono

$$N_{int}(\varepsilon_{inf}, \varepsilon_{sup}) = \sum_i \sigma_{cls}[\varepsilon(y_i; \varepsilon_{inf}, \varepsilon_{sup})]\, A_i + \sum_j \sigma_{acc}[\varepsilon(y_j; \varepsilon_{inf}, \varepsilon_{sup})]\, A_j \tag{6}$$

$$M_{int}(\varepsilon_{inf}, \varepsilon_{sup}) = \sum_i \sigma_{cls}[\varepsilon(y_i; \varepsilon_{inf}, \varepsilon_{sup})]\, y_i\, A_i + \sum_j \sigma_{acc}[\varepsilon(y_j; \varepsilon_{inf}, \varepsilon_{sup})]\, y_j\, A_j \tag{7}$$

dove sono stati esplicitati i contributi delle armature j. In generale si possono definire quanti materiali si desiderano, con i relativi legami costitutivi.

La condizione di rottura vincola una delle due incognite. Infatti, se si ha una rottura lato calcestruzzo, la deformazione superiore deve essere pari alla deformazione ultima del calcestruzzo $\varepsilon_{sup} = \varepsilon_{cu}$. In caso di rottura lato acciaio la deformazione del lembo inferiore è pari alla deformazione ultima dell'acciaio $\varepsilon_{inf} = \varepsilon_{su}$. Questo comporta che la flessione nelle sezioni analizzate è sempre presa *positiva*, ovvero che tende le fibre inferiori. Nel caso di flessione negativa il superiore si "inverte" con l'inferiore. Una volta fissata una delle due incognite, ε_{sup} o ε_{inf} in caso si ha rispettivamente una rottura lato calcestruzzo o lato acciaio, la seconda è trovata grazie all'equazione di equilibrio alla traslazione assiale $N_{int}(\varepsilon_{inf}, \varepsilon_{sup}) = N$, dove N è lo sforzo assiale a cui è soggetta la sezione.

Ponendo, per esempio, che la rottura avvenga lato calcestruzzo e quindi, per un momento agente positivo (che comprime le fibre superiori), si ha $\varepsilon_{sup} = \varepsilon_{cu}$ dove si ricorda che ha un valore negativo

perché di *accorciamento*. Allora in questo particolare caso si vuole trovare ε_{inf} invertendo l'equazione di equilibrio

$$N_{int}(\varepsilon_{inf}, \varepsilon_{sup} = \varepsilon_{cu}) = N \tag{8}$$

Dato che si tratta di un'equazione non-lineare per invertirla si necessita di un metodo di soluzione numerico adatto. In questo caso si utilizza l'algoritmo della *bisezione*. Tale algoritmo consiste nel trovare un range di variabilità del parametro in esame (in questo esempio ε_{inf}) e successivamente riduce tale intervallo in base alla valutazione dell'equazione in esame nei punti estremi ed in quello intermedio. Spiegare meglio in una appendice, oppure mettere un riferimento Pro. Come intervallo iniziale per la variabile ricercata si prende

$$\varepsilon_{cu} \leq \varepsilon_{inf} \leq \varepsilon_{su} \tag{9}$$

```
eps_infN = eps_cu; %limite inferiore dell'intervallo (negativo: accorciamento)
eps_infP = eps_su; %limite superiore dell'intervallo (positivo: allungamento)
```

in quanto, in qualunque caso, non può essere fuori da tale intervallo (altrimenti saremmo in una condizione post-critica e quindi non più in un punto di rottura). Il valore dello sforzo normale agente N (negativo di compressione) sarà sicuramente compreso tra i valori dello sforzo normale interno N_{int} calcolato nei due punti estremi dell'intervallo

$$N_{int}(\varepsilon_{inf,N}, \varepsilon_{sup} = \varepsilon_{cu}) \leq N \leq N_{int}(\varepsilon_{inf,P}, \varepsilon_{sup} = \varepsilon_{cu}) \tag{10}$$

La procedura della *bisezione* consiste nel determinare lo sforzo normale in un punto intermedio tra i due

$$\varepsilon_{inf,M} = \frac{\varepsilon_{inf,N} + \varepsilon_{inf,P}}{2} \tag{11}$$

ed aggiornare l'intervallo prendendo questo punto intermedio come estremo e quindi *dimezzare* l'intervallo di ricerca[4]. La decisione su quale dei due estremi (inferiore o superiore) deve assumere il valore medio dipende dal valore della funzione da risolvere nel punto medio. In particolare se $N_{int}(\varepsilon_{inf,M}, \varepsilon_{sup} = \varepsilon_{cu}) < N$ allora vuol dire che il valore medio è il nuovo limite inferiore ($\varepsilon_{inf,N} = \varepsilon_{inf,M}$) poiché questo permette di mantenere lo sforzo normale agente N all'interno dell'intervallo di ricerca. Al contrario, se $N_{int}(\varepsilon_{inf,M}, \varepsilon_{sup} = \varepsilon_{cu}) > N$ allora il valore medio è il nuovo limite superiore ($\varepsilon_{inf,P} = \varepsilon_{inf,M}$).

inserire una immagine di uno step di bisezione per rendere tutto più chiaro (anche con il riferimento della sezione affianco)

Tale procedura di dimezzamento dell'intervallo viene ripetuta finché i due estremi non convergono ad un valore, che risulta essere il risultato cercato. Nel caso in cui lo strumento di calcolo possiede una precisione infinita e vengono eseguite infinite iterazioni di dimezzamento, i due estremi dovrebbero convergere in un valore unico (se il problema è ben posto). Tuttavia questo non è tecnicamente possibile attraverso strumenti di calcolo odierni. La procedura iterativa si conclude quando vengono passati determinati test di convergenza. In particolare un test di convergenza sulla risultante dello sforzo normale e sul valore degli estremi di ricerca

$$\left|N_{int}(\varepsilon_{inf,M}, \varepsilon_{sup} = \varepsilon_{cu}) - N\right| < |\text{tol} \cdot N| \qquad \left|\varepsilon_{inf,P} - \varepsilon_{inf,N}\right| < \left|\text{tol} \cdot \varepsilon_{inf,P}\right| \tag{12}$$

dove "tol" è il valore di tolleranza in termini percentuali. Nel caso in cui si ha la rottura lato acciaio tutte le considerazioni rimangono uguali, tranne che per il fatto che ad essere fissato è ε_{inf} e la ricerca della bisezione avviene per ε_{sup}.

Una volta trovati i valori di ε_{inf} e ε_{sup} gli sforzi interni N_{int} e M_{int} li ricava grazie alle equazioni riferimento, mentre le altre quantità sono pari a

$$\chi = \frac{\varepsilon_{inf} - \varepsilon_{sup}}{H} \qquad y_n = \frac{\varepsilon_{sup}}{\varepsilon_{sup} - \varepsilon_{inf}} H \tag{13}$$

1.5 Grafico momento-curvatura

[4]Proprio per il fatto che ad ogni iterazione l'intervallo di ricerca si dimezza questo metodo è chiamato *metodo di bisezione*.

2 Sezioni in calcestruzzo armato

In questa sezione vengono confrontati i risultati della routine *MATLAB* "`MchiNstrisce`" con altri software di analisi delle sezioni:

- VcaSLU
- EC2
- FranchinEEtoolbox
- SAP2000
- SeRes

EC2 funziona malissimo Fare una "description" dei vari programmi utilizzati dicendo da dove vengono e chi li ha fatti?

Le caratteristiche dei materiali sono:

Calcestruzzo:	$f_c = 40\ MPa$	$\varepsilon_{c0} = 0.0020$	$\varepsilon_{cu} = 0.0035$
Acciaio:	$f_y = 400\ MPa$	$\varepsilon_y = 0.0020$	$\varepsilon_{su} = 0.03$

inserire una immagine dei legami costitutivi

Le sezioni presentate in questa sezione sono da considerarsi ideali ed ideate solamente con lo scopo di testare i vari le routine in base ai risultati di altri programmi di calcolo. Si è ben consapevoli che, in alcune occasioni, determinate sezioni non sarebbero realizzabili a causa dei vincoli imposti dalle normative tecniche locali.

2.1 Pilastro 30x30cm

inserire l'immagine della sezione con TikZ

2.2 Pilastro 30x50cm

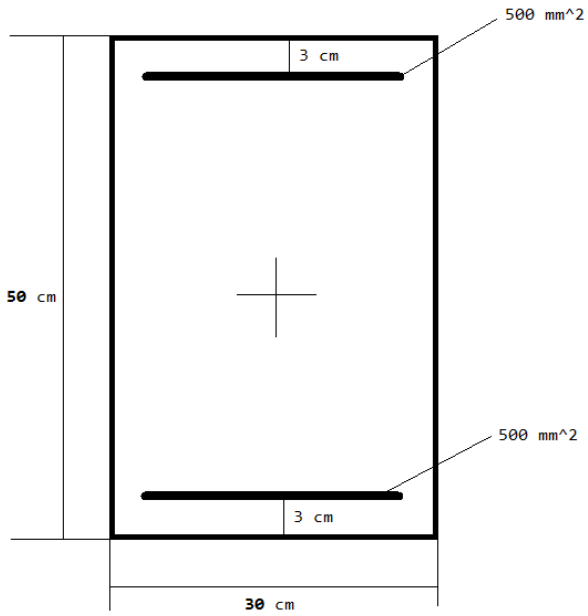


Figura 4: Pilastro $30 \times 50\ cm$.

migliorare immagine

	M_u [MNm]	ϕ_u [1/m]	y_u [m]	M_y [MNm]	ϕ_y [1/m]	y_y [m]
VcaSLU	0.092	0.1134	0.029	0.088	0.005	-
EC2	0.092	-	0.028	-	-	-
FranchinEEtb	0.093	0.135	~0.03	0.089	0.006	~0.09
FranchinEEtb(anal)	0.092	0.1336	0.021	0.089	0.005	0.081
`MchiNstrisce` (1)[5]	0.092	0.1207	0.029	0.082	0.005	0.088
`MchiNstrisce` (2)[6]	0.092	0.1206	-	0.090	0.005	-
`MchiN`(anal) [7]						

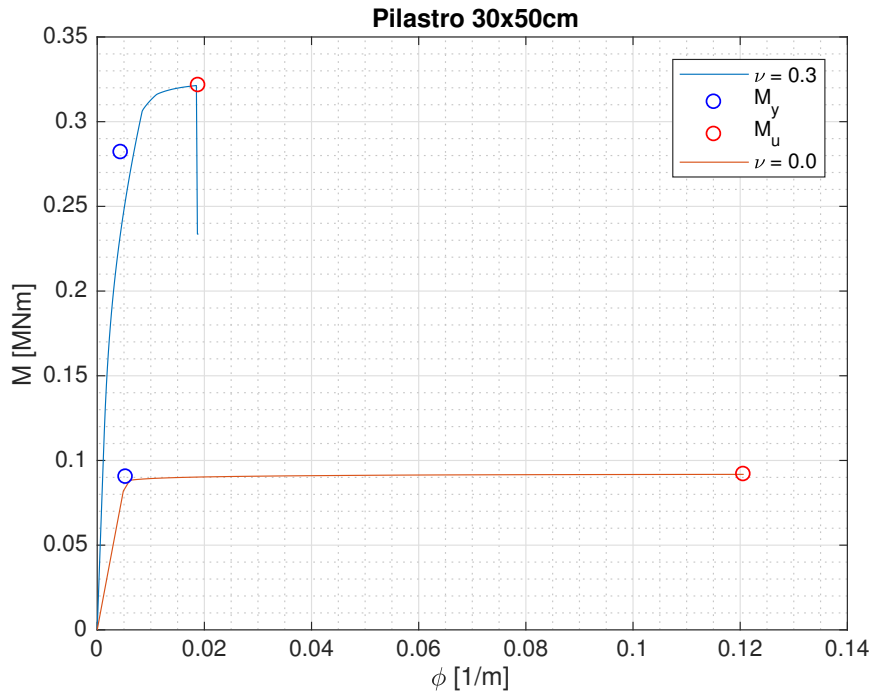
Tabella 1: Risultati per il pilastro a sezione 30×50 *cm* con $\nu = 0.0$.

	M_u [MNm]	ϕ_u [1/m]	y_u [m]	M_y [MNm]	ϕ_y [1/m]	y_y [m]
VcaSLU	0.322	0.0178	0.185	0.307	0.008	-
FranchinEEtb	0.400	0.021	~0.21	0.383	0.009	~0.27
FranchinEEtb(anal)	0.403	0.0187	0.188	0.414	0.008	0.204
`MchiNstrisce` (1)	0.321	0.0189	0.185	0.296	0.008	0.239
`MchiNstrisce` (2)	0.321	0.0185	-	0.281	0.004	-
`MchiN`(anal)						

Tabella 2: Risultati per il pilastro a sezione 30×50 *cm* con $\nu = 0.3$.

L'errore massimo è del 10% sul valore di M_y, mentre ϕ_y presenta una variabilità anche del 50%. Il valore del momento ultimo M_u e della profondità dell'asse neutro nello stato ultimo y_u sono sostanzialmente identici. Le differenze nella definizione del momento e della curvatura di snervamento sono imputabili al fatto che la routine `findPoints2` trova il punto di snervamento facendo un fit della curva con un modello bilineare. Nell'immagine seguente sono riportate le curve $M-\phi$ per la sezione in esame.

Le soluzioni analitiche trovate tramite il foglio di Excel del Professor Franchin ("`FranchinEEtoolbox.xlsx`") sono riferite ad una sezione con armatura semplice (solo uno strato di armatura inferiore). il momento ultimo è minore di quello di snervamento per il caso $\nu = 0.3$?



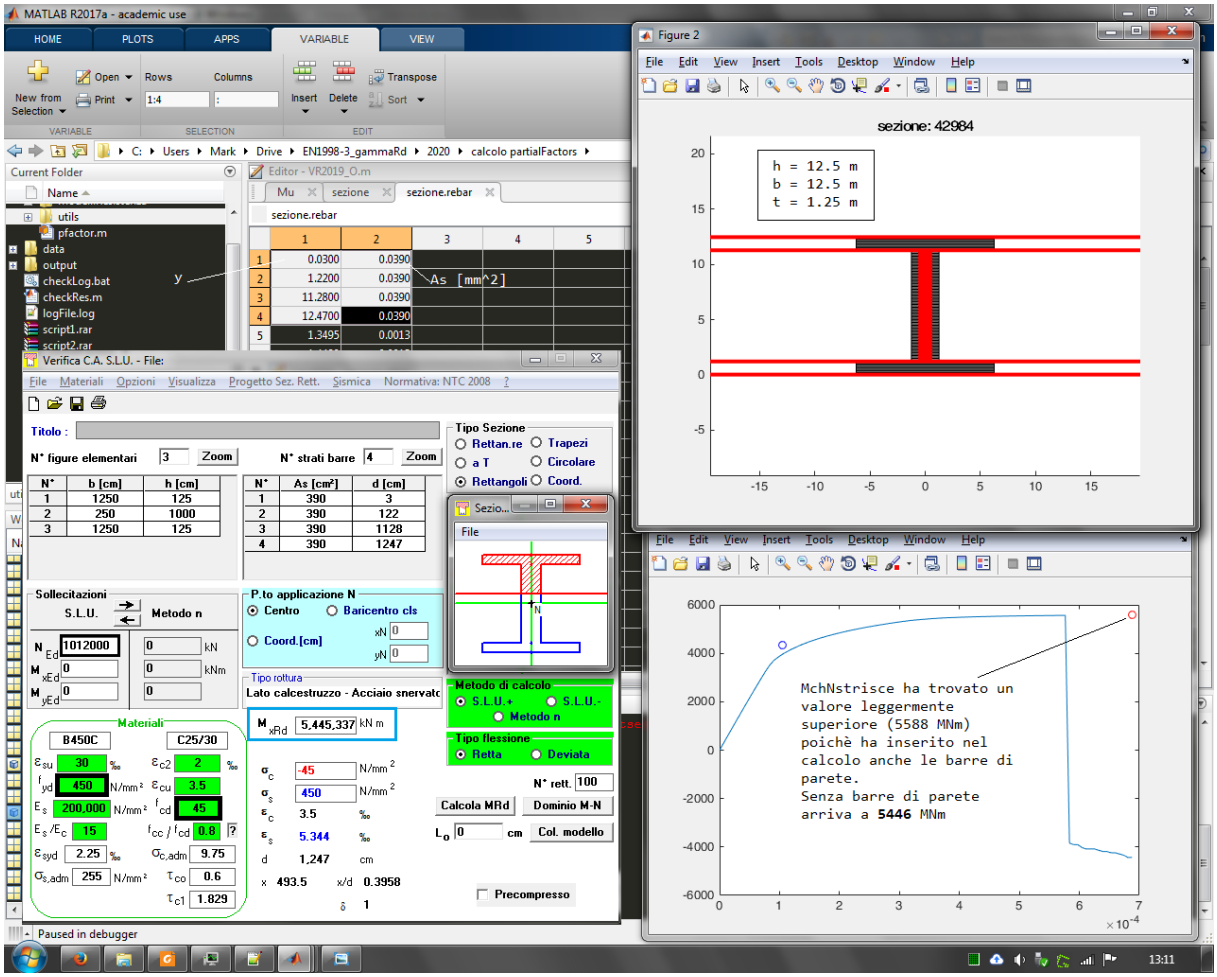
[5] `yieldingPoints` e `ultimatePoints`
[6] `findPoints2`
[7] Analitico

2.3 Palo 120cm

PARE CHE CI SIA UNA DISCREPANZA TRA VCASLU E MCHIN (vedere i calcoli per il palo del Gatteo)

Forse c'è un problema nel calcolo delle sezioni circolari...

2.4 Pila 1250x1250x125



3 Sezioni in acciaio

In questa sezione vengono confrontati i risultati della routine *MATLAB* "`MchiNstrisceSteel`" con i dati dei profilati ed altri software di analisi delle sezioni:

- VcaSLU
- EC2
- SAP2000
- SeRes

A Esempio di utilizzo della classe

Di seguito è riportato il codice per calcolare la sezione del pilastro 30x50 cm.

```
%testa le procedure MchiN

addpath(genpath(pwd));


%% Pilastro 30x50cm

clc

%parametri
H  = 0.5;     %[m]
B  = 0.3;     %[m]
c  = 0.03;    %[m] (copriferro)
fc = 30;      %[MPa]
fy = 400;     %[MPa]
As = 500/1e6; %[m^2] (area di uno strato di armatura)
nu = 0.3;    %[-] (compressione normalizzata)
Nr = fc*H*B;  %[MN] (sforzo normale massimo)

disp('Pilastro 30x50cm');

sezione = MchiNstrisce(H/1000,@ParabRett,@ElastPlast);
sezione.defineMaterials(fc,fy,0.002,0.0035,0.002,0.06);
sezione.addLayer(H,B,B);
sezione.addRebar(c,As); %armatura inferiore
sezione.addRebar(H-c,As); %armatura superiore
sezione.initgeo;

N = - nu * Nr; %negativo: compressione

[M,phi,yn] = sezione.ultimatePoint(N);
disp('ultimatePoint:');
disp(['M_u    = ' num2str(M)]);
disp(['phi_u  = ' num2str(phi)]);
disp(['yn_u   = ' num2str(yn)]);
[M,phi,yn] = sezione.yieldingPoint(N);
disp('yieldingPoint:');
disp(['M_y    = ' num2str(M)]);
disp(['phi_y  = ' num2str(phi)]);
disp(['yn_y   = ' num2str(yn)]);

[My,chiy,Mu,chiu] = sezione.findPoints2(N,1);
disp('findPoints2:');
disp(['M_y    = ' num2str(My)]);
disp(['phi_y  = ' num2str(chiy)]);
disp(['M_u    = ' num2str(Mu)]);
disp(['phi_u  = ' num2str(chiu)]);
```

B Programma *MATLAB*

B.1 MchiNstrisce.m

```
% Questa classe e' basata sul codice SeReS_strisce

classdef MchiNstrisce < handle
    properties
        hmax %altezza massima del layer
        layer %matrice dove ogni riga e' un layer e le colonne sono: y, h, b_bot, b_top
        rebar %matrice dove ogni riga e' una barra e le colonne sono: y, As, mat_id
%         materials %struttura che contiene le funzioni sigma(eps) dei materiali (ogni riga e' un materiale differente)
        legameCLS %legame sigma(eps) del CLS
        legameAcc %legame sigma(eps) dell'acciaio
        %proprieta' aggiuntive:
        layersProp %ogni riga e' un layer, le colonne sono: Area, ygl
        %proprieta' dei materiali:
        fc %resistenza calcestruzzo
        fy %resistenza acciaio
        epscy, epscu %deformazione di snervamento e ultima del CLS
        epssy, epssu %deformazione di snervamento e ultima dell'acciaio
        %costanti della sezione:
        Area    %Area della Sezione
        yg      %coordinata del baricentro
        Ix      %Momento di inerzia secondo l'asse orizzontale
        H       %altezza totale della sezione
    end
    methods
        function obj = MchiNstrisce(hmax,CLS,Acc)
            %costruttore della classe
            obj.hmax = hmax;
            obj.legameCLS = CLS;
            obj.legameAcc = Acc;
        end

        function addLayer(obj,h,b,t)
            %aggiunge tanti layer alti massimo h_max fino a creare la patch
            %desiderata
            y = 0;
            for i=1:size(obj.layer,1); y = y + obj.layer(i,2); end
            sum = 0;
            while (h - (sum+obj.hmax) > 0)
                bj = b + (t-b)/(h)*(sum);
                tj = b + (t-b)/(h)*(sum+obj.hmax);
                if size(obj.layer,1) > 0; obj.layer = [obj.layer; (obj.layer(size(obj.layer,1),1)+obj.layer(size(obj.layer,1),2)) obj.hmax bj tj];
                else; obj.layer = [obj.layer; 0 obj.hmax b tj]; end %aggiunge il primo layer
                sum = sum + obj.hmax;
            end
            bj = b + (t-b)/(h)*(sum);
            if size(obj.layer,1) > 0; obj.layer = [obj.layer; (obj.layer(size(obj.layer,1),1)+obj.layer(size(obj.layer,1),2)) h-sum bj t];
            else; obj.layer = [obj.layer; 0 h b t]; end %aggiunge il primo layer
        end

        function addRebar(obj,y,As)
            %aggiunge uno strato di armature ordinarie
            obj.rebar = [obj.rebar; y As];

        end

        function defineMaterials(obj, fc, fy, eps_cy, eps_cu, eps_sy, eps_su)
            %aggiunge le proprieta' dei materiali
            if nargin < 4
                obj.fc = -abs(fc); %negativo: compressione
                obj.fy = fy;
                obj.epscy = -0.002; %negativo: accorciamento
                obj.epscu = -0.0035;
```

```
63                  obj.epssy = 0.002;
64                  obj.epssu = 0.03; %il massimo a cui sono arrivato e' il 6% (le NTC mi pare
                        prescrivano l'1%)
65              else
66                  obj.fc = -abs(fc);
67                  obj.fy = fy;
68                  obj.epscy = -abs(eps_cy);
69                  obj.epscu = -abs(eps_cu);
70                  obj.epssy = eps_sy;
71                  obj.epssu = eps_su;
72              end
73          end
74
75          function initgeo(obj)
76              %calcola la geometria della sezione
77              %si prende in considerazione il fatto che sia composta tutta dello stesso
                    materiale (in realta' dovrei SEMPRE omogeneizzare i layer [per ogni layer un
                    coeff. di omogenizzazione n])
78              obj.Area = 0;
79              obj.Ix = 0;
80              Sx = 0;
81              for i=1:size(obj.layer,1)
82                  %1  2  3  4
83                  %y, h, b, t, mat_id
84                  Areal = 0.5 * obj.layer(i,2) * (obj.layer(i,3) + obj.layer(i,4));
85                  ygl   = obj.layer(i,1) + obj.layer(i,2)/3 * (1+obj.layer(i,4)/(obj.layer(i,3)+
                        obj.layer(i,4)));
86                  obj.Area = obj.Area + Areal;
87                  obj.layersProp = [obj.layersProp; Areal ygl];
88                  Sx = Sx + obj.layer(i,3) * obj.layer(i,2) * (obj.layer(i,2)/2 + obj.layer(i,1)
                        ) + 0.5*obj.layer(i,2)*(obj.layer(i,4)-obj.layer(i,3))*(2/3.*obj.layer(i
                        ,2) + obj.layer(i,1));
89                  obj.Ix = obj.Ix + 1/36*(obj.layer(i,2))^3*(2*obj.layer(i,3)+obj.layer(i,4)) +
                        obj.layer(i,2)*(obj.layer(i,3)*(1/2*obj.layer(i,2)+obj.layer(i,1))^2 +
                        1/2*(obj.layer(i,4)-obj.layer(i,3))*(2/3*obj.layer(i,2)+obj.layer(i,1))^2)
                        ;
90              end
91              obj.yg = Sx / obj.Area;
92              obj.Ix = obj.Ix - obj.Area * (obj.yg)^2;
93              obj.H = sum(obj.layer(:,2));
94          end
95
96          function [N,M] = setStrain(obj,eps_b,eps_t)
97              %calcola N e M impostata la deformazione al lembo inferiore
98              %(eps_b) e al lembo superiore (eps_t)
99
100             %forze nei layers:
101             Flayer = obj.legameCLS((eps_t-eps_b)*obj.layersProp(:,2)/obj.H+eps_b, obj.epscy,
                    obj.epscu,obj.fc) .* obj.layersProp(:,1);
102             %forze nelle barre:
103             Frebar = obj.legameAcc((eps_t-eps_b)*obj.rebar(:,1)/obj.H+eps_b, obj.epssy,obj.
                    epssu,obj.fy) .* obj.rebar(:,2);
104
105             %contributo del calcestruzzo
106             N = sum(Flayer);
107             M = sum(Flayer .* obj.layersProp(:,2));
108             %contrivuto dell'acciaio
109             N = N + sum(Frebar);
110             M = M + sum(Frebar .* obj.rebar(:,1));
111
112             M = M - N * obj.yg; %aggiunge il momento dovuto alla forza normale
113         end
114
115         function [M,phi,yn] = yieldingPoint(obj, N)
116             %calcola il punto di primo snervamento dato uno sforzo normale N
117             % DA MIGLIORARE L'ALGORITMO!!!!!!! (vedi la procedura usata per
118             % "ultimatePoint")
119
120             if N < obj.fc*obj.Area; error('la sezione non puo' resistere a tale compressione')
                    ; end
```

```
121             if N > obj.fy*sum(obj.rebar(:,2)); error('la sezione non puo' resistere a tale
                    trazione'); end
122
123             % fissato lo eps_inf = epssy cambia eps_top fino a trovare N corretto
124             eps_inf = obj.epssy;
125
126             %trova un range per fare la bisezione
127             count = 0;
128             eps_supP = obj.epssy; %l'ho scelto io (l'N iniziale puo' essere solamente minore)
129             eps_supN = obj.epssy; %valore Negativo
130             while obj.setStrain(eps_inf,eps_supN) > N
131                 eps_supN = eps_supN - 0.001;
132                 count = count + 1;
133                 if count > 1000; error('MchiNstrisce non e' arrivato a convergenza'); end
134             end
135
136             %fa la bisezione
137             err = [100 100 100];
138             count = 0;
139             while ((abs(err(2)) > abs(0.005 * N)) )%|| (abs(eps_supP-eps_supN) > abs(0.0001*
                    eps_supN))) %+0.000001 serve per non creare errori per N=0
140                 err = [(obj.setStrain(eps_inf,eps_supP) - N) (obj.setStrain(eps_inf,(eps_supP+
                        eps_supN)/2) - N) (obj.setStrain(eps_inf,eps_supN) - N)];
141                 if err(1)*err(2) < 0
142                     %la risultante e' tra err(1) e err(2) (eps_supP e' corretto)
143                     eps_supN = (eps_supP+eps_supN)/2;
144
145                 else
146                     %la risultante e' tra err(2) e err(3) (eps_supN e' corretto)
147                     eps_supP = (eps_supP+eps_supN)/2;
148                 end
149                 count = count + 1;
150                 if count > 1000; error('MchiNstrisce non e' arrivato a convergenza'); end
151             end
152
153             eps_sup = (eps_supP+eps_supN)/2;
154
155             [~,M] = obj.setStrain(eps_inf,eps_sup);
156             phi = (eps_inf-eps_sup)/obj.H; %compressione: negativa (la fibra superiore e'
                    compressa)
157             yn = -eps_sup/(eps_inf-eps_sup) * obj.H;
158             M = -M; %convenzione italiana
159         end
160
161         function [M,phi,yn] = ultimatePoint(obj, N)
162             %calcola il momento e la curvatura ultimi
163             %il valore di N e' considerato negativo di compressione
164
165             if N < obj.fc*obj.Area; error('la sezione non puo' resistere a tale compressione')
                    ; end
166             if N > obj.fy*sum(obj.rebar(:,2)); error('la sezione non puo' resistere a tale
                    trazione'); end
167
168             %calcola lo sforzo normale interno della sezione nello stato di
169             %rottura limite (armature a eps_su e calcestruzzo a eps_cu) e
170             %se e' comunque minore dello sforzo agente N (quindi se la
171             %sezione e' piu' compressa dello sforzo normale agente) vuol dire
172             %che l'asse neutro deve "alzarsi" e quindi deve aumentare la
173             %epsilon superiore (aumentare perche' prima era negativa:
174             %-0.35%) Questo ultimo caso porta ad una rottura lato acciaio
175             %(ovvero che e' fissa eps_inf e la bisezione si fa su eps_sup)
176             if obj.setStrain(obj.epssu,obj.epscu) > N
177                 %si ha una rottura lato calcestruzzo, quindi e' tenuto fisso
178                 %eps_sup = eps_cu
179                 rottura_acciaio = 0;
180                 eps_sup = obj.epscu; %deformazione ultima del calcestruzzo (negativa: di
                        accorciamento)
181             else
182                 %si ha una rottura lato acciaio, quindi e' tenuto fisso
183                 %eps_inf = eps_su
```

```
184                 rottura_acciaio = 1;
185                 eps_inf = obj.epssu; %deformazione ultima dell'accaio (positiva: di
                        allungamento)
186             end
187
188 %             if rottura_acciaio; warning('rottura lato acciaio'); end
189 %             if rottura_acciaio; logFile('MchiN::ultimatePoint: rottura lato acciaio'); end
190
191
192             %fa la bisezione
193             err = [100 100 100]; %variabile di errore per la bisezione: [negativo, medio,
                    positivo]
194             count = 0;
195             if rottura_acciaio == 0
196                 %rottura del CLS:
197                 eps_infP = obj.epssu; %deformazione ultima dell'acciaio (non puo' essere
                        maggiore)
198                 eps_infN = obj.epscu; %deformazione ultima del calcestruzzo (non puo' essere
                        minore)
199                 while ((abs(err(2)) > abs(0.0001 * N + 1e-18)) || (abs(eps_infP-eps_infN) >
                        abs(0.0001*eps_infN))) %+1e-18 serve per non creare errori per N=0)
200                     eps_infM = (eps_infP + eps_infN) / 2; %epsilon medio tra i due estremi (
                            risparmio tre calcoli per ogni iterazione al costo di aggiungere una
                            variabile)
201                     err = [(obj.setStrain(eps_infN,eps_sup) - N) (obj.setStrain(eps_infM,
                            eps_sup) - N) (obj.setStrain(eps_infP,eps_sup) - N)];
202                     if err(2) > 0
203                         %la risultante e' tra err(1) e err(2) (eps_infN e' corretto)
204                         eps_infP = eps_infM;
205                     else
206                         %la risultante e' tra err(2) e err(3) (eps_infP e' corretto)
207                         eps_infN = eps_infM;
208                     end
209                     count = count + 1;
210                     if count > 10000; error('MchiNstrisce non e' arrivato a convergenza'); end
211                 end
212                 eps_inf = eps_infM;
213             else
214                 %rottura dell'acciaio:
215                 eps_supP = 0; %potrebbe essere maggiore, potrebbe essere la deformazione di
                        snervamento dell'acciaio
216                 eps_supN = obj.epscu; %deformazione ultima del calcestruzzo (non puo' essere
                        minore)
217                 while ((abs(err(2)) > abs(0.0001 * N + 1e-18)) || (abs(eps_supP-eps_supN) >
                        abs(0.0001*eps_supN))) %+1e-18 serve per non creare errori per N=0)
218                     eps_supM = (eps_supP + eps_supN) / 2; %epsilon medio tra i due estremi (
                            risparmio tre calcoli per ogni iterazione al costo di aggiungere una
                            variabile)
219                     err = [(obj.setStrain(eps_inf,eps_supN) - N) (obj.setStrain(eps_inf,
                            eps_supM) - N) (obj.setStrain(eps_inf,eps_supP) - N)];
220                     if err(2) > 0
221                         %la risultante e' tra err(1) e err(2) (eps_supN e' corretto)
222                         eps_supP = eps_supM;
223                     else
224                         %la risultante e' tra err(2) e err(3) (eps_supP e' corretto)
225                         eps_supN = eps_supM;
226                     end
227                     count = count + 1;
228                     if count > 9000
229                         disp('sta per fallire');
230                     end
231                     if count > 10000; error('MchiNstrisce non e' arrivato a convergenza'); end
232                 end
233                 eps_sup = eps_supM;
234             end
235
236             [~,M] = obj.setStrain(eps_inf,eps_sup);
237             phi = (eps_inf-eps_sup)/obj.H; %compressione: negativa (la fibra superiore e'
                    compressa) IN REALTa' e' IL CONTRARIO (NEGATIVA)
238             yn = -eps_sup/(eps_inf-eps_sup) * obj.H; %yn definito partendo all'"alto"
```

```
239             M = -M; %convenzione italiana
240         end
241
242
243         function eps = deformazione(obj,eps0,chi,y)
244             %funzione di epsilon: def(eps0,chi,y) = eps0 + chi*(y-yg)
245             eps = eps0 + chi*(y-obj.yg);
246         end
247
248         function [f] = F(obj,eps0,chi,N0,M0)
249             %calcola le funzioni
250             %FUNZIONE AUSILIARIA
251             [N,M]  = obj.Fint(eps0,chi);
252             f(1,1) = N - N0;
253             f(2,1) = M - M0;
254         end
255
256         function [N,M] = Fint(obj,eps0,chi)
257             %restituisce le forze interne della sezione dato un profilo di
258             %deformazione (definito da eps0 e chi)
259
260             %-----------------------------------------------------------
261             %NUOVA FORMULAZIONE SENZA -yg NELLA FUNZIONE "deformazione"
262             %-----------------------------------------------------------
263
264             N = sum(obj.legameCLS(obj.deformazione(eps0,chi,obj.layersProp(:,2)), obj.epscy,
                    obj.epscu,obj.fc) .* obj.layersProp(:,1)) + ...
265                 sum(obj.legameAcc(obj.deformazione(eps0,chi,obj.rebar(:,1)), obj.epssy,obj.
                        epssu,obj.fy) .* obj.rebar(:,2));
266             M = sum(obj.legameCLS(obj.deformazione(eps0,chi,obj.layersProp(:,2)), obj.epscy,
                    obj.epscu,obj.fc) .* (obj.layersProp(:,2)-obj.yg) .* obj.layersProp(:,1)) +
                    ...
267                 sum(obj.legameAcc(obj.deformazione(eps0,chi,obj.rebar(:,1)), obj.epssy,obj.
                        epssu,obj.fy) .* (obj.rebar(:,1)-obj.yg) .* obj.rebar(:,2));
268         end
269
270         function [M,chi] = curvaMchi3(obj,N,chi_lim)
271             % calcola la curva M-chi-N
272             %    inizialmente calcola i punti della curva con il metodo di
273             %    Newton-Raphson e se fallisce calcola con la bisezione
274             %    Con N=0 da problemi numerici, deve essere sempre <0
275
276             if nargin < 3; chi_max = (obj.epssu-obj.epscu)/obj.H; else; chi_max = chi_lim; end
277
278             N_punti = 100; %numero punti della curva
279             delta = 1.e-12; %variazione per il calcolo numerico della derivata parziale
280             tol = 1.e-8; %tolleranza per la convergenza (per il metodo di Newton)
281
282             chi = linspace(0,chi_max,N_punti);
283             M   = zeros(1,N_punti);
284
285             for i=2:(N_punti) %parte da 2 perche' cosi' M(0) = 0 (altrimenti non lo e' per
                    problemi numerici...)
286                 err   = 1; %variabile di errore (per il test di convergenza)
287                 eps0  = 0; %punto iniziale
288                 count = 0; %contatore che monitora il caso in cui va in loop infinito
289                 bisez = 0; %controlla se non bisogna ricorrere alla bisezione
290                 Nint  = 0;
291                 deltaNint = 100;
292                 while (err > tol) || (deltaNint > 0.001*abs(N))
293                     Nint_old = Nint;
294                     [Nint,~] = obj.Fint(eps0,chi(i));
295                     deltaNint = abs(Nint-Nint_old);
296                     derivata = (obj.Fint(eps0+delta,chi(i))-Nint)/delta;
297                     eps0_old = eps0;
298                     eps0 = eps0 + (1/derivata) * (N - Nint);
299                     if eps0 ~= 0; err = abs((eps0 - eps0_old)/eps0); else; err = abs(eps0 -
                            eps0_old); end
300                     count = count + 1;
301                     if count > 100 || derivata == 0
```

```
302 %                         warning('MchiN::curvaMchi2: Newton-Raphson non va bene, calcolo con
        la bisezione...');
303                           logFile('MchiN::curvaMchi2:␣Newton-Raphson␣non␣va␣bene,␣calcolo␣con␣la
                                  ␣bisezione...');
304                           bisez = 1;
305                           break;
306                       end
307                   end
308                   if bisez
309                       %cerca il range di laboro per la bisezione:
310                       %fa la ricerca attraverso il "grid search"
311                       for j=0:4
312                           temp = linspace(-obj.epssu,obj.epssu,(2^j)*10);
313                           Nint = zeros(size(temp));
314                           for k=1:size(temp,2)
315                               Nint(k) = obj.Fint(temp(k),chi(i));
316                           end
317                           [~,i_min] = min(Nint);
318                           [~,i_max] = max(Nint);
319                           if (Nint(i_min)<N && Nint(i_max)>N); break; end %ha trovato il range
320                           %altrimenti continua finche' non e' discretizzato a
321                           %sufficienza
322                       end
323                       if (Nint(i_min)>N || Nint(i_max)<N)
324                           %se alla fine del for non ha ancora trovato il
325                           %range allora non ci sono speranze neanche per la
326                           %bisezione
327 %                           error('MchiN::curvaMchi2::bisezione: ERRORE nella ricerca del range
        per la bisezione');
328                           %provare ad usare il "gradient descend" dal punto
329                           %temp(i_min)
330                           %IL TEST SULLA SEZIONE CIRCOLARE FALLISCE A QUESTO
331                           %PUNTO PER \nu=0.3
332                           M(i) = 0;
333                           return;
334                       end
335                       eps0_min = temp(i_min);
336                       eps0_max = temp(i_max);
337
338                       %esegue la bisezione:
339                       count = 0;
340                       Ntot = [1 1 1];
341                       tol_bis = 0.00001; %tolleranza per la bisezione
342                       while (norm(Ntot) > (tol_bis * abs(N))) || (abs(abs(eps0_max)-abs(eps0_min
                              )) > (tol_bis * abs(eps0_min)))
343                           eps0 = (eps0_min + eps0_max) / 2; %valore medio
344                           Ntot(1) = obj.Fint(eps0_min,chi(i)) - N;
345                           Ntot(2) = obj.Fint(eps0,chi(i)) - N;
346                           Ntot(3) = obj.Fint(eps0_max,chi(i)) - N;
347                           if (Ntot(1)*Ntot(2)) < 0
348                               eps0_max = eps0;
349                           else
350                               eps0_min = eps0;
351                           end
352                           count = count + 1;
353                           if count > 100
354                               logFile('MchiN::curvaMchi2::bisezione:␣ERRORE␣convergenza');
355                               error('MchiN::curvaMchi2::bisezione:␣non␣e' arrivato a convergenza
                                      ');
356                           end
357                       end
358                   end
359                   [~,M(i)] = obj.Fint(eps0,chi(i));
360               end
361           end
362
363           function [My,chiy,Mu,chiu] = findPoints(obj, N, ~)
364               %trova i punti di snervamento e ultimo della sezione dalla
365               %curva curvatura-momento.
366
```

```
367             [~,chi_max] = obj.ultimatePoint(N); %METODO PER STABILIZZARE I RISULTATI NUMERICI
368             [M,chi] = obj.curvaMchi3(N,chi_max); %calcola la curva "grezza"
369 
370             %trova il punto ultimo come il punto in cui la curva comincia
371             %ad avere la derivata negativa (i materiali non hanno tratti
372             %degradanti a tangente negativa):
373 %             i = find(diff(M) < 0 ,1); %tronca quando comincia a decrescere CERTE VOLTE
    FALLISCE PER PROBLEMI NUMERICI ALL'INIZIO
374             [~,i] = max(M); %tronca quando raggiunge il massimo
375             Mu   = M(i);
376             chiu = chi(i);
377             %elimina la parte di curva dopo il punto ultimo:
378             M   = M(1:i);
379             chi = chi(1:i);
380 
381             %trova il punto di snervamento bilinearizzando la curva:
382             P0 = [chiu/3 3/4*Mu]; %parametri iniziali di prova
383             model = @(P,x) (P(2)/P(1)*x).*heaviside(P(1)-x) + ((Mu-P(2))/(chiu-P(1))*(x-P(1))+
                    P(2)).*heaviside(x-P(1)); %crea il modello da fittare
384             lb = [0 0]; %lower bound (non ci possono essere valori negativi)
385             ub = [chiu Mu]; %upper bound (il punto di snervamento non puo' essere maggiore di
                    quello ultimo)
386             options = optimoptions('lsqcurvefit','Display','off');
387             Pfit = lsqcurvefit(model,P0,chi,M,lb,ub,options); %fitta il modello minimizzando i
                     quadrati
388             chiy = Pfit(1);
389             My   = Pfit(2);
390 
391             %controlla i risultati:
392 
393             %comandi per i test:
394             if nargin > 2
395                 plot(chi,M); hold on;
396                 plot(linspace(0,chiu,100),model(Pfit,linspace(0,chiu,100)),'k--');
397                 plot(chiy,My,'bo');
398                 plot(chiu,Mu,'ro');
399             end
400 
401         end
402 
403         function [My,chiy,Mu,chiu] = findPoints2(obj, N, ~)
404             %trova i punti trovando prima chi_u e poi discretizzando la
405             %curva in un numero fisso di punti
406 
407             [Mu,chiu] = obj.ultimatePoint(N);
408             [M,chi] = obj.curvaMchi3(N,chiu); %calcola la curva
409 
410             %trova il punto di snervamento bilinearizzando la curva:
411             P0 = [chiu/3 3/4*Mu]; %parametri iniziali di prova
412             model = @(P,x) (P(2)/P(1)*x).*heaviside(P(1)-x) + ((Mu-P(2))/(chiu-P(1))*(x-P(1))+
                    P(2)).*heaviside(x-P(1)); %crea il modello da fittare
413             lb = [0 0]; %lower bound (non ci possono essere valori negativi)
414             ub = [chiu Mu]; %upper bound (il punto di snervamento non puo' essere maggiore di
                    quello ultimo)
415             options = optimoptions('lsqcurvefit','Display','off');
416             Pfit = lsqcurvefit(model,P0,chi,M,lb,ub,options); %fitta il modello minimizzando i
                     quadrati
417             chiy = Pfit(1);
418             My   = Pfit(2);
419 
420             %DEBUGGING
421             if nargin > 2
422                 plot(chi,M); hold on;
423                 plot(chiy,My,'ob');
424                 plot(chiu,Mu,'or');
425             end
426 
427         end
428 
429         function out = test(obj,eps0,chi)
```

```
430             %esegue dei test sulla sezione
431
432             N = zeros(size(eps0));
433             M = zeros(size(eps0));
434
435             for i=1:size(eps0,2)
436                 [N(i),M(i)] = obj.Fint(eps0(1,i),chi);
437             end
438
439             out(:,1) = N;
440             out(:,2) = M;
441
442         end
443
444         function initRect(obj,h,b,As1,As2,c)
445             %inizializza una sezione rettangolare
446             if nargin < 6; c = 0.03; end
447             obj.addLayer(h,b,b); %crea la sezione in CLS
448             obj.addRebar(c,As1);
449             obj.addRebar(h-c,As2);
450
451             obj.initgeo;
452         end
453
454         function initRect2(obj,h,b,rhotot,rhoc_rhotot,db,c)
455             %inizializza una sezione rettangolare (con anche l'armatura di parete)
456             if nargin < 7; c = 0.03; end
457             Ac = b*h; %area calcestruzzo
458             Ab = pi/4 * db^2; %area della barra
459             np = max(round((h-2*c)/0.3-1), 0); %calcola il numero di armature di parete (non
                    ci puo' essere in interferro maggiore di 30 cm tra esse) [maggiore uguale a
                    zero..]
460             n2 = round(Ac*rhotot/Ab * rhoc_rhotot); %calcola il numero di armature superiori (
                    compresse)
461             n1 = round(Ac*rhotot/Ab - n2 - 2*np);  %calcola il numero di armature inferiori (
                    tese)
462
463             obj.addLayer(h,b,b); %crea la sezione in CLS
464             obj.addRebar(c,n1*Ab); %crea l'armatura inferiore
465             obj.addRebar(h-c,n2*Ab); %crea l'armatura superiore
466             %aggiunge le armature di parete:
467             if np > 0
468                 interferro_laterale = (h-2*c)/(np+1);
469                 y = c + interferro_laterale;
470                 for i=1:np
471                     obj.addRebar(y,2*Ab);
472                     y = y + interferro_laterale;
473                 end
474             end
475
476             obj.initgeo; %calcola la geometria della sezione
477         end
478
479         function initCirc(obj,D,db,rhotot,c)
480             %inizializza una sezione circolare
481             if nargin < 6; c = 0.03; end
482             AreaBarra = pi/4 * db^2;
483             numeroBarre = round(rhotot * (D^2)/(db^2));
484
485             hs = D/50;
486
487             for y=linspace(0,D-hs,D/hs)
488                 thetab = acos(1 - 2/D*y);
489                 b = D*sin(thetab);
490                 thetat = acos(1 - 2/D*(y+hs));
491                 t = D*sin(thetat);
492                 obj.addLayer(hs,real(b),real(t));
493             end
494             for theta=linspace(0,pi,(numeroBarre+1)/2)
495                 y = D/2 - ((D-2*c)/2)*cos(theta);
```

```
496                 obj.addRebar(y,2*AreaBarra);
497             end
498 
499             obj.initgeo;
500         end
501 
502         function initHollow(obj,h,b,s,db,rhotot,c)
503             %inizializza una sezione rettangolare cava
504             if nargin < 7; c = 0.03; end
505             AreaBarra = pi/4 * db^2;
506             nBarre = round(rhotot*(8*s*(h+b-2*s))/(pi*db^2));
507             interferro = 4*(h+b-2*s)/nBarre;
508             nBarreParete = round(4*(h-2*s)/interferro); %round(8/pi * ((h-2*s)*s)/(db^2));
509             nBarreStrati = round((nBarre-nBarreParete)/4);
510 
511             %crea la sezione in CLS
512             obj.addLayer(s,b,b);
513             obj.addLayer(h-2*s,2*s,2*s);
514             obj.addLayer(s,b,b);
515             %aggiunge gli strati orizzontali
516             obj.addRebar(c,AreaBarra*nBarreStrati);
517             obj.addRebar(s-c,AreaBarra*nBarreStrati);
518             obj.addRebar(h-s+c,AreaBarra*nBarreStrati);
519             obj.addRebar(h-c,AreaBarra*nBarreStrati);
520             %aggiunge gli strati verticali
521             interferro_parete = (h-2*s)/(nBarreParete/4 + 1);
522             y = s + interferro_parete;
523             for i=1:(nBarreParete/4)
524 %                 obj.addRebar(s+(h-2*s)*(i/(nBarreParete/4)),4*AreaBarra);
525                 obj.addRebar(y,4*AreaBarra);
526                 y = y + interferro_parete;
527             end
528 
529             obj.initgeo;
530         end
531 
532         function plotSection(obj)
533             %plotta la sezione
534 
535             for i=1:size(obj.layer,1)
536                 %plotta i layers
537                 y = obj.layer(i,1); %ordinata del lembo inferiore del layer
538                 h = obj.layer(i,2); %altezza del layer
539                 b = obj.layer(i,3); %larghezza di base del layer
540                 t = obj.layer(i,4); %larghezza superiore del layer
541                 Xp = [-b/2 b/2 t/2 -t/2];
542                 Yp = [y y y+h y+h];
543                 patch(Xp,Yp,[0.8 0.8 0.8]);
544                 hold on;
545             end
546 
547             for i=1:size(obj.rebar,1)
548                 %plotta le barre
549                 y = obj.rebar(i,1);
550                 As = obj.rebar(i,2);
551 %                 line([-As/2*1000 As/2*1000],[y y],'Color','red');
552                 plot([-As/2*1000 As/2*1000],[y y],'r','LineWidth',3);
553             end
554 
555                         hold off;
556 %             text(0,obj.H/2,num2str(size(obj.rebar,1)));
557             axis equal;
558         end
559 
560 
561     end
562 
563 end
564 
565 
```

```
566
567 % ChangeLog:
568 % ----------
569 %   13/01/2021: Ho modificato le funzioni "ultimatePoint" eliminando la ricerca iniziale del
        range e "Fint" eliminando il -yg nel calcolo dellal deformazione
570 %   16/01/2021: Ho eliminato "findEquilibrium" e "curvaMchi" che non erano usate da tempo
571 %   17/01/2021: Ho inserito la funzione "curvaMchi3" che migliora "curvaMchi2"
572 %
```

B.2 Legami costitutivi

B.2.1 Calcestruzzo

Listing 1: legame costitutivo parabola-rettangolo per il calcestruzzo armato (`ParabRett.m`)

```
1  function sigma = ParabRett(eps,epsy,epsu,smax)
2      %legame costitutivo CLS NTC2018
3      %eps e' negativo di compressione
4
5  %     epsy = -0.0020;
6  %     epsu = -0.0035;
7  %     smax = -30; %MPa
8
9      sigma = zeros(size(eps));
10     for i=1:size(eps)
11         if (eps(i) > 0); sigma(i) = 0; %non resistente a trazione
12         elseif (eps(i) >= epsy);        sigma(i) = smax * (2*eps(i)/epsy - (eps(i)*eps(i))/(
               epsy*epsy));
13         elseif (eps(i) >= epsu);        sigma(i) = smax;
14         elseif (eps(i) < epsu);    sigma(i) = 0.0;
15         end
16     end
17 end
```

Listing 2: legame costitutivo parabola-rettangolo per il calcestruzzo armato con la definizione anche della resistenza a trazione (`ParabRett2.m`)

B.2.2 Acciaio

Listing 3: legame costitutivo elasto-plastico perfetto per l'acciaio (`ElaspPlast.m`)

```
1  function sigma = ElastPlast(eps,epsy,epsu,smax)
2      %legame costitutivo elastoplastico perfetto
3
4  %     %acciaio B450C (NTC2018):
5  %     epsy = 0.002;
6  %     epsu = 0.010;
7  %     smax = 391; %MPa
8
9      sigma = zeros(size(eps));
10     for i=1:size(eps)
11         if (eps(i) <= -epsu);           sigma(i) = 0.0;
12         elseif (eps(i) <= -epsy);       sigma(i) = -smax;
13         elseif (eps(i) <= epsy);        sigma(i) = eps(i) * (smax/epsy);
14         elseif (eps(i) <= epsu);        sigma(i) = smax;
15         elseif (eps(i) > epsu);         sigma(i) = 0.0;
16         end
17     end
18 end
```