Qui verranno spiegate le function più importanti e più utilizzate che sono raccolte nel file *myTesi.m.*

theMission

theMission produce come output il costo totale della missione e l'oggetto *mission* in cui sono raccolti tutti i parametri di ogni sottomissione che vengono monitorati: la lista dei detriti, la lista dei detriti avanzati, il costo in ΔV, il costo totale e le masse. In input prende:

- *t_vet*, un vettore di istanti temporali
- *ddata*, i parametri orbitali dei detriti che sono stati importati dal file Excel
- *mission*, la missione di riferimento, di cui si vogliono monitorare le sottomissioni
- *c*, dove sono raccolte alcune costanti

Compaiono poi altre funzioni più semplici, tra cui:

- *massaMission*, che calcola le masse iniziali delle sottomissioni
- *prezzo_massa*, che calcola il costo in MEUR a seconda della massa iniziale
- *costo_attesa*, che calcola il costo in MEUR in base al tempo impiegato dalle sottomissioni.

```
function [tc,mission] = theMission(t_vet,ddata,mission,c)

 [dV]=myTesi.estimatedV(ddata,t_vet,c.start_ep,0,mission.n,mission.list);

  %mission.left = true(1,c.n);
  %mission.list = [];
  kk = 0;
  detriti_raccolti = 0; finale = 0;
  for j=1:length(mission.pat)
      if mission.feas(j)
              kk = kk+1;
           i2 = sum(mission.n(1:j)-1);
           i1 = i2-(mission.n(j)-1)+1;

          dVsub = dV(i1:i2); %<-- ora dV è un semplice vettore, non un tensore.

          %pat     = mission.pat{j};
          %mission.list   = cat(2,mission.list,pat);
          %mission.left(pat) = false;
          %indices = sub2ind(size(dVsub),pat(1:end-1),pat(2:end),1:mission.n(j)-1);
          %  mission.n(j)             = length(pat);

          mission.estdV{j} = dVsub;

          mission.total_cost(j) = sum(mission.estdV{j});
          mission.costo_medio_min(j) = mean(mission.estdV{j});
          mission.estM(j) = massaMission(c.Mdry, c.Mde, mission.estdV{j}, c.G0, c.Isp);
          mission.massMEUR(j) = myTesi.prezzo_massa(c.Mdry, mission.estM(j));

          finale = finale + sum( t_vet(detriti_raccolti+ (1:mission.n(j))));
          detriti_raccolti = detriti_raccolti + mission.n(j);
```

```
            mission.timeMEUR(j) = myTesi.costo_attesa(c.start_ep, c.stop_ep, finale +c.start_ep);
        end
    end

tc = sum(mission.timeMEUR+mission.massMEUR);
mission.TC = tc;
end
```

estimatedV

Questa funzione serve per il calcolo dei ΔV. In input prende i parametri orbitali dei dtriti, importati dal file Excel, un vettore che rappresenta gli istanti in cui iniziano i trasferimenti orbitali, il giorno di inizio della competizione, la durata del trasferimento (che può essere unico per tutti e quindi essere uno scalare o variabile e quindi un vettore).

varargin può essere vuoto o meno:

- se *varargin* è vuoto, *dV* sarà una matrice tridimensionale, poichè saranno stati calcolati i costi dei trasferimenti di ogni detrito con ogni altro detrito
- se *varargin* non è vuoto, *dV* sarà un vettore, poichè saranno calcolati solo i trasferimenti tra i detriti che fanno parte della sequenza di missione che viene passata in input.

Questa è l'unica differenza tra i due casi; per il resto vengono applicate le formule Shen per il calcolo della propagazione dei parametri orbitali.

```
function [dV] = estimatedV(ddata,dt,start_ep,trasf_dur,varargin)
[~,ref_epoch,a,e,i,Om,om,~] = deal(ddata{:});

% *) SE VARARGIN è vuoto:
% viene calcolato il dV di tutti gli n-debris con tutti gli n-debris
% è possibile che "trasf_dur" sia unico per tutti (uno scalare) oppure
% variabile nel corso della missione. In quest'ultimo caso "trasf_dur"
% deve avere lo stesso numero di elementi di "dt".
% [trasf_dur(i) = durata del trasferimento orbitale i-esimo]
% [dt(i)        = istante in cui è iniziato il trasferimento i-esimo]
if isempty(varargin)
    if (size(dt,1) >1) || (size(dt,2) >1)
        dt = reshape(dt,1,1,[]);
    end

    if numel(trasf_dur) == numel(dt)
        trasf_dur = reshape(trasf_dur,1,1,[]);
    end
else
    % *) SE VARARGIN non è vuoto:
    % viene calcolato il dV prendendo come riferimento la lista della missione
    % servono alcune informazioni indispensabili:
    N = varargin{1};  %<-- (vettore mission.n)
    list = varargin{2}; %<-- (vettore mission.list)
    if size(dt,2)>1
        dt = reshape(dt,[],1);
```

```
    end
    trasf_dur = dt(2:end); %<--  durata dei trasferimenti,
                           % sono presenti anche le pause (+30days)
    dt = cumsum(dt);    %<-- istanti temporali dei trasferimenti e degli arrivi.
    dt(end) = [];       %<-- l'ultimo istante temporale è l'arrivo sul 123° detrito
                        % non serve.
    andata = list(1:end-1);
    arrivo = list(2:end);
end
j2 = 1.08262668e-3;
rE = 6378137;       %[m]
mi = 398600.4418e9; %[m^3/s^2]

Omega_dot= @(a,e,i) -3/2.*sqrt(mi./a.^3).*j2.*cos(i)./(1-e.^2).^2.*(rE./a).^2;
Om_dot = Omega_dot(a,e,i);
omega_dot= @(a,e,i) 3/4*sqrt(mi./a.^3)*j2.*(5.*cos(i).^2-1)./((1-e.^2).^2).*((rE./a).^2);
om_dot = omega_dot(a,e,i);


if isempty(varargin)
Om0_dot = (Om_dot+Om_dot')/2;
OM = Om + Om_dot.*(start_ep+dt-ref_epoch)*86400;
OM = rem(OM, 2*pi);
differenza = OM-permute(OM, [2 1 3]);
i0 = (i+i')/2;
a0 = (a+a')/2;
W  = om + om_dot.*(start_ep+dt-ref_epoch)*86400;
W  = rem(W, 2*pi);
dex = e.*cos(W) -e.*cos(permute(W,[2 1 3]));
dey = e.*sin(W) -e.*sin(permute(W,[2 1 3]));
V0 = sqrt(mi./a0);
y = V0.*(a-a')/2./a0;
z = (i-i').*V0;
else
    Om0_dot = (Om_dot(andata)+ Om_dot(arrivo))/2;

    OM1 = Om(andata) + Om_dot(andata) .*(start_ep+dt-ref_epoch(andata))*86400;
    OM2 = Om(arrivo) + Om_dot(arrivo) .*(start_ep+dt-ref_epoch(arrivo))*86400;
    differenza = OM2-OM1;
    i0 = mean(i(andata)+i(arrivo))/2;
    a0 = mean(a(andata)+a(arrivo))/2;
    V0 = sqrt(mi./a0);
    W1 = om(andata) + om_dot(andata) .*(start_ep+dt-ref_epoch(andata))*86400;
    W2 = om(arrivo) + om_dot(arrivo) .*(start_ep+dt-ref_epoch(arrivo))*86400;
    W1 = rem(W1, 2*pi); W2  = rem(W2, 2*pi);

    dex = e(arrivo).*cos(W2) -e(andata).*cos(W1);
    dey = e(arrivo).*sin(W2) -e(andata).*sin(W1);
    y = V0.*(a(arrivo)-a(andata))/2./a0;
    z = (i(arrivo)-i(andata)).*V0;
end


dVe = 1/2.*V0.*sqrt((dey).^2+(dex).^2);
```

```
differenza = rem(differenza,2*pi);
differenza(differenza> pi) = differenza(differenza> pi)-2*pi;
differenza(differenza<-pi) = differenza(differenza<-pi)+2*pi;
x = (differenza).*sin(i0).*V0;
m =        7*Om0_dot.*sin(i0).* (trasf_dur-5) * 24*3600;
n = Om0_dot.*tan(i0).*sin(i0).* (trasf_dur-5) * 24*3600;
sx = (2*x+m.*y+n.*z)./(4+m.^2+n.^2)./x;
sy = (2*m.*x-(4+n.^2).*y+m.*n.*z)./(8+2*m.^2+2*n.^2)./y;
sz = (2*n.*x+m.*n.*y-(4+m.^2).*z)./(8+2*m.^2+2*n.^2)./z;
dx =  m.*y.*sy+n.*z.*sz;
dVa = sqrt((sx.*x).^2+(sy.*y).^2+(sz.*z).^2);
dVb = sqrt((x-sx.*x-dx).^2+(y+sy.*y).^2+(z+sz.*z).^2);
dV  = sqrt((0.5*dVe).^2+dVa.^2)+sqrt((0.5*dVe).^2+dVb.^2);

if ~isempty(varargin)
    % elimino dal vettore dV quegli elementi a cavallo tra la fine e l'inizio
    % della sotto-missione successiva.
    id = cumsum(N); id(id==sum(N)) = [];
    dV(id) = [];
end
end
```

creaConstraints

Dal momento che è stata imposta una convenzione algebrica focalizzata specificamente sugli intervalli temporali di ogni trasferimento e delle rispettive pause, è necessario automatizzare la creazione dei limiti (intorni) entro cui ogni incognita può variare. In presenza di un trasferimento orbitale i giorni a disposizione vanno da 5 a 30; invece, gli intervalli temporali tra una sottomissione e la successiva partono da un minimo di 30 giorni a un valore arbitrario (l'utente può specificarlo in *vals).* Unica eccezione è il valore della prima incognita che sancisce l'inizio di tutta la missione. A seconda della partizione con cui i detriti sono stati raccolti, otterremo:

- limite inferiore (*lb*): 0 [5 5 ..5] 30 [5 5 ..5] 30 [5 5 ..5]..
- limite superiore (*ub*): vals(2) [30 30 ..30] vals(1) [30 30 ..30] vals(1) [30 30 ..30]..
- *A* e *B* termini della disequzione del tipo $Ax \leq B$, dove essendo x costituito da tutti gli intervalli temporali è necessario che la somma di questi non superi gli 8 anni; *A* si trasforma pertanto in un vettore di coefficienti unitari per ottenere la durata complessiva di missione e *B* si identifica con la durata in giorni di 8 anni.
- *t0* vettore di guess: vals(4) [vals(5) ..] vals(3) [vals(5)....] vals(3) ...

Questa function viene utilizzata solo nella fase di ottimizzazione con **fmincon**.

```
function [lb,ub,t0,Adis,Bdis] = creaConstraints(vals,c,nlist)
    % lower bound: 0 [5 5 ..5] 30 [5 5 ..5] 30 [5 5 ..5];
    % lower bound è un vettore con (1+122) elementi
    % il 1° sancisce il momento a partire da start_ep, della 1° missione
    % i restanti 122 sono gli intervalli tra un detrito e il successivo
    N = sum(nlist);
    lb = repmat(5,N,1); % all'inizio tutti 5
    lb(cumsum(nlist) +1) = 30; % i primi di ogni sottomissione sono 30
```

```
    lb(1) = -0.1; % anticipo della prima sottomissione

    % upper bound: 100 [30 30 ..30] 60 [30 30 ..30] 60 [30 30 ..30];
    ub = repmat(30,N,1); % i primi sono 30
    ub(cumsum(nlist) +1) = vals(1);  %  massima pausa imponibile
    ub(1) = vals(2); % ritardo della prima sottomissione

    % La somma complessiva degli intervalli deve soddisfare:
    % somma(dt) <= (stop_ep-start_ep)
    Adis = ones(1,N); % A x < B; x sono le durate,
                      % sommate tra di loro devono essere minori di 8 anni
    Bdis = -c.start_ep+c.stop_ep;

    % Valore di guess iniziale, t0_vet
    t0 = vals(5)*ones(N,1);
    t0(cumsum(nlist) +1) = vals(3); %~31, sopra la pausa minima di 30
    t0(1) = vals(4);  %~0
    lb = lb(1:N); ub=ub(1:N); t0=t0(1:N);
end
```