```
clc, clear, close all
global finestra tab
```

Nel file *myTesi.m* sono riunite per comodità di utilizzo la maggior parte delle funzioni usate

```
f = myTesi;

% importazione dei dati
data = importdata("gtoc9_debris.xlsx");
data = data.data.gtoc9_debris(:,1:end-1);
ddata = mat2cell(data,size(data,1),ones(1,size(data,2)));
clear data
```

Inizializzazione di alcune costanti:

```
c.start_ep = 23467; % inizio MJD2000
c.stop_ep  = 26419; % fine MJD2000
c.n  = 123;         % numero detriti da rimuovere
c.Mdry  = 2000;     % dry mass (2000 kg)
c.Mde = 30;         % massa del kit di de-orbit
c.G0 = 9.80665;     % accelerazione di gravità
c.Isp = 340;        % impulso specifico
```

- Suddivido la linea temporale di 8 anni di operatività in un numero finito di frames (300).

```
frames = 300;
t = linspace(c.start_ep,c.stop_ep,frames);
```

- Creo un grafo orientato in cui memorizzare come legami percorribili solamente i passaggi da detrito a detrito con un costo (in termini di ΔV) inferiore ad una determinata soglia, scelta in modo arbitrario. Nota: una soglia troppo bassa rende il grafo poveramente connesso, cosa da evitare. Una soglia troppo alta coincide con un blando filtro e il risparmio economico che ne consegue è compromesso.

```
g=digraph;   % qui i grafi sono vuoti
DV = digraph;
soglia = 370; % 370 m/s, soglia del deltaV nella ricerca
```

- Creo una matrice *tab* per tenere traccia dei detriti coinvolti nello studio. Ogni detrito è definito in modo univoco in base a due parametri: il suo identificativo (da 1 a 123) e dal frame temporale a cui fa riferimento il passaggio orbitale (da 1 a 300).

```
tab = zeros(c.n,frames);
```

- Decido di ripetere la valutazione dei trasferimenti orbitali: caso i = 1 $\longrightarrow$ 5 giorni di trasferimento + 5 giorni di rendez-vous; caso i = 2 $\longrightarrow$ 15 giorni di trasferimento + 5 di rendez-vous; caso i = 3 $\longrightarrow$ 25 giorni di trasferimento + 5 di rendez-vous.

```
for i=1:3
tr = i*(t(2)-c.start_ep)+5.1*(i==0);
```

Richiamo la funzione **estimatedV** dall'oggetto '*f*' in cui sono contenute tutte le funzioni del foglio *myTesi.m*. Tramite **estimatedV** ottengo rapidamente una matrice 123x123x300 i cui valori esprimono il costo in ΔV. **estimatedV** ha già valutato i parametri orbitali corrispondenti ad ogni istante temporale e ha rimosso 5 giorni di rendez-vous dal ΔT messo a disposizione, identificato con '*tr*'.

```
dV = f.estimatedV(ddata,t-c.start_ep,c.start_ep,tr);
```

Solo i costi inferiori alla soglia specificata sono stati considerati. Un generico elemento *(i,j,k)* appartenente alla matrice *dV* inferiore alla soglia equivale a dire che il trasferimento orbitale all'istante t(k) con durata tr (incluso rendez-vous) dal detrito i al detrito j è possibile o risulta conveniente.

```
sotto = find(dV(:,:,1:end-i)<soglia);
[rig,col,tem] = ind2sub([c.n c.n size(dV,3)-i],sotto);
```

Tutti i detriti in partenza o in arrivo vengono contrassegnati modificando il valore presente nella matrice *tab*.

```
tab(rig+(tem-1)*c.n)=tab(rig+(tem-1)*c.n)+i;
%tab(col+(tem+i-1)*c.n)=tab(col+(tem+i-1)*c.n)+i;
```

Poichè nella sintassi di matlab le funzioni relative ai grafi prevedono l'uso di stringhe, le informazioni numeriche sono state convertite in stringhe.

```
andata = (rig+tem/1000)+""; % '1.001' equivale a detrito 1 frame 1
arrivo = (col+(tem+i)/1000)+""; % '4.005' equivale a detrito 4 frame 5
```

Aggiungo il legame al grafo orientato, specificando che tale legame sussiste tra il detrito (nodo) e all'istante contenuti in *andata* e il detrito e all'istante contenuti in *arrivo*. Nel grafo *g* memorizzo il legame assegnando come peso un valore possibile tra 1,2,3 in base alla durata del trasferimento orbitale, mentre nel grafo *DV* il peso del legame coincide con il costo in ΔV (m/s).

```
g = addedge(g,andata,arrivo,i);  % aggiungo il legame con 'addedge(...)'
DV = addedge(DV,andata,arrivo,dV(sotto));
end
```

```
copertura0 = nnz(tab)/(c.n*frames)
```

```
copertura0 = 0.9758
```

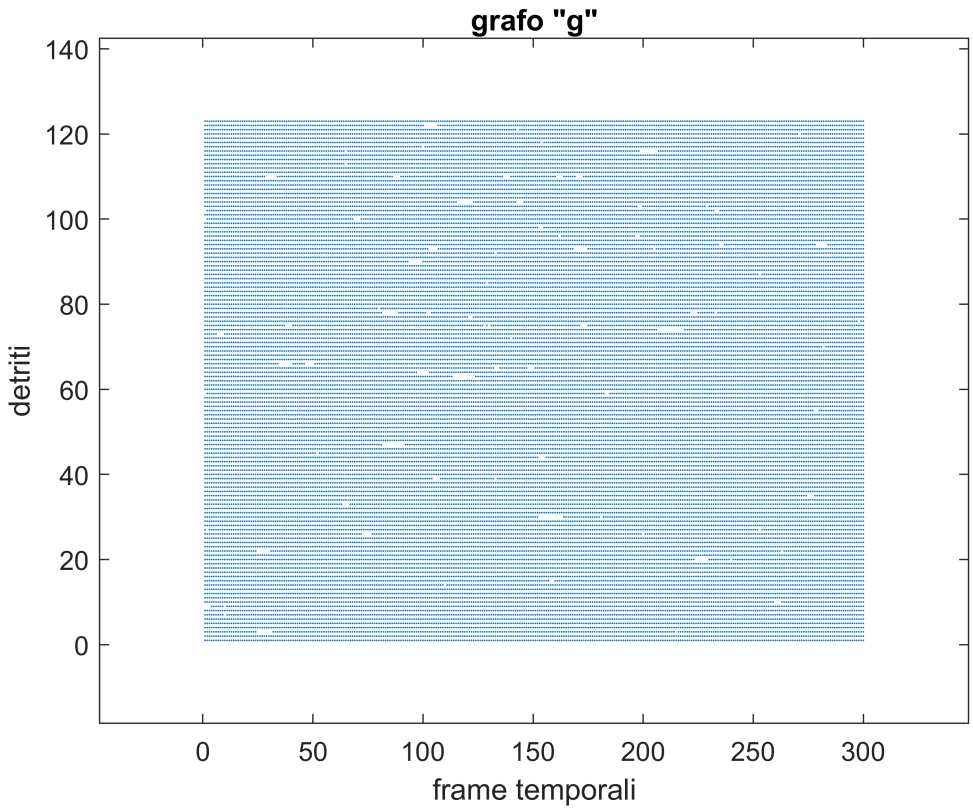
```
copertura1 = nnz(dV<soglia)/numel(dV)
```

```
copertura1 = 0.0326
```

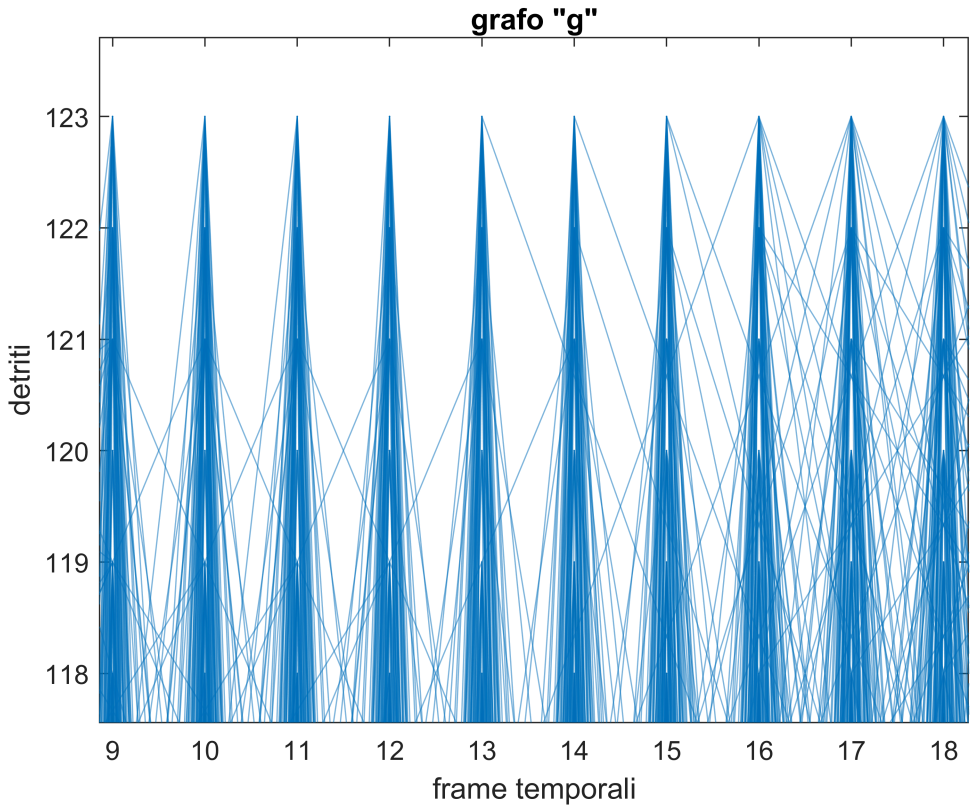
```
y = uint8(double(string(g.Nodes.Name)));
x = uint8(rem(double(string(g.Nodes.Name)),1)*1000);

% Rappresentazione del grafo:
```

```
nn = g.Nodes.Name;
ids = arrayfun(@(x) floor(eval(x)), string(nn));
tims = arrayfun(@(x) rem(eval(x),1)*1000, string(nn));
figure
p = plot(g,'ShowArrows',"off",'XData',tims,'YData',ids,'LineStyle',"none",'Marker','.');
xlabel('frame temporali'), ylabel('detriti'), title('grafo "g"')
```




```
p.LineStyle = "-";
```



Creazione preliminare delle sequenze

Questa sezione di codice è stata volontariamente commentata poichè la sua esecuzione richiede del tempo e viene lasciata solo per uso informativo. La principale finalità, come spiegato in precedenza, è la ricerca di sequenze percorribili tra detriti in accordo con le restrizioni temporali. Nel codice si fa uso del toolbox dei grafi di Matlab (vengono usate funzioni come successors(), etc.).

La ricerca viene svolta prendendo più filoni temporali (che sono chiamati *tatoo* nel codice), nei quali viene memorizzato un possibile percorso. La struttura del grafo, essendo dotata di una bassa copertura, impedisce la creazione di un filone senza interruzioni. Pertanto, ogni filone viene ampliato a più riprese, svolgendo ricerche nel grafo indagando i nodi successivi a partire da un nodo di partenza. Tra un'interruzione e l'altra, che può essere dovuta al fatto che il nodo non possiede connessioni sotto al valore di soglia (ΔV), vi si dovrà identificare e far coincidere l'inizio e la fine di due diverse sottofasi della missione spaziale generale. Al termine della ricerca ogni filone può essere elaborato e modificato per giungere ad una soluzione del problema, poichè opera indipendentemente dagli altri filone.

Ogni sottofase prevede un massimo di 500 iterazioni, modficabili dall'utente. I filoni analizzati in contemporanea sono 20.

```
eseguiCodice = true;
titolo = "prova"+".mat";

switch eseguiCodice
    case 1
```

```
 mostraVid = true;
 iterations = 500;
 ntr = 20;

 i = 0;  trasla = zeros(1,ntr);
 tatoo = cell(ntr,1);
 storia = repmat("",ntr,1);
 finestra = 3;
 continua = 1;
 debris_target = 105;
```

La ricerca termina quando almeno uno dei filoni colleziona 105 detriti o più. Per alimentare ogni filone viene creata una variabile temporanea di nome *tr* (abbreviazione di travellers), che possiede molte caratteristiche.

```
while continua
%classif.L = 35;

% trasla = min(trasla,frames-finestra);
close all, clear tr video
tr(ntr) = struct;
video = cell(1,ntr);
```

Come prima operazione la variabile *tr* di ogni filone viene inizializzata assegnandole un nodo di partenza, in modo che:

- il detrito di quel nodo non sia già presente nel filone relativo
- il nodo sia distante almeno 30 giorni (cioè +3 frames) rispetto all'ultimo nodo nella sequenza del filone
- il nodo possieda legami con altri nodi

Se ci sono più nodi con queste caratteristiche l'assegnazione è casuale.Se nessun nodo rispetta queste caratteristiche l'assegnazione si concentra su un gruppo di nodi appartenenti a frames temporali successivi.

```
for k = 1:ntr
    nonvabene = 1;
    while nonvabene
         if (trasla(k)>frames-finestra)
         nonvabene = 0;
         tr(k).ko = true;
         else
         tr(k).ko = false;
         end

         if ~tr(k).ko && nonvabene
```

Qui avviene l'assegnazione preliminare del nodo grazie alla matrice tab creata nella sezione precedente.

```
        [starter,timess] = find(tab(:,(1:finestra)+trasla(k)));
        tr(k).mover = trasla(k);
```

Qui si rimuovono i nodi i cui detriti sono stati già presi (nella prima iterazione tutti i detriti vanno bene)

```
        [new_starter,ia]=setdiff(starter,tatoo{k});
```

```
        nonvabene = isempty(new_starter);
         end

        if nonvabene
         trasla(k) = trasla(k) + finestra;
        end
    end
```

Se un gruppo di nodi ha soddisfatto tutti i criteri elencati prima allora sono eligibili come nodi di inizializzazione. Di loro ci interessano l'identificativo (nel codice *new_starter*) e il frame temporale a cui appartiene (nel codice *new_timess*).

```
    if ~tr(k).ko
    new_timess = timess(ia);
    piscina = getId(new_starter,new_timess+trasla(k));
    id = randperm(length(new_starter),1); % la scelta del nodo di inizializzazione è casuale
```

Ogni traveller dopo essere stato inizializzato con un nodo di partenza tiene traccia di:

1. la sequenza che verrà costruita più avanti e che incrementerà il filone
2. un insieme di nodi alternativi per costruire la lista
3. il numero più alto di detriti raccolti senza interruzioni

```
    tr(k).pox = cell(1,0);
    tr(k).pox{1} = piscina;

    tr(k).idlist(1) = tr(k).pox{1}(id);
    tr(k).list = getNode(tr(k).idlist(1));
    tr(k).nbest = 0;
    end

     if mostraVid
     video{k} = text(0,-k,"",'Color','k'); hold on
     text(-0.5,-k,num2str(length(tatoo{k})),'Color','g','Horizon','right');
     end
end

% classif.list   = repmat("",classif.L,1);
% classif.idlist = repmat("",classif.L,1);
% classif.score  = zeros(classif.L,1);

best = 0; bestlist = [];
axis([-0.1 3 -ntr-1 0])
```

Il traveller inizializzato viene incrementato e svolge le ricerche cercando nodi compatibili con quello di partenza.

```
for j = 1:iterations
    for k = 1:ntr
      if ~tr(k).ko
       tr(k) = esplora(g,tr(k),tatoo{k});
```

La funzione *esplora* incrementa la lista del travellers e prende in input il grafo *g*, creato nella sezione precedente. La funzione si può visualizzare in fondo alla pagina.

Dopo ogni iterazione la sequenza potrebbe essere stata incrementata, quindi si conta la sua lunghezza e la si confronta con la sequenza più lunga trovata. In caso si trovi una lunghezza maggiore rispetto alle iterazioni precedenti questa diventa la nuova sequenza principe e sarà quella che incrementerà il filone.

```
ll = length(tr(k).list);

        if ll>tr(k).nbest
             tr(k).best = tr(k).list;
             tr(k).idbest = tr(k).idlist;
             tr(k).nbest = ll;
        end

%         if any(ll > classif.score)
%
%            % controllo se la sequenza esiste già, ignoro i tempi. se non esiste la
%            % aggiungo. se esiste già controllo quale delle due ha il marker
%            % temporale più basso.
%            seqnodi = strjoin(string(tr(k).list),"_");
%            confronto = strcmp(classif.list,seqnodi);
%            if ~any(confronto)
%              classif.idlist(end+1) = strjoin(tr(k).idlist,"_");
%              classif.list(end+1)   = seqnodi;
%              [score,ordine] = sort( [classif.score; ll],'descend');
%              classif.score  = score(1:classif.L);
%              classif.list   =   classif.list(ordine(1:end-1));
%              classif.idlist = classif.idlist(ordine(1:end-1));
%
%            elseif any(confronto)
%               avv1 = classif.idlist(confronto);
%               t1 = nuovo(avv1,0);
%               avv2 = strjoin(tr(k).idlist,"_");
%               t2 = nuovo(avv2,0);
%
%               if t2<t1
%                 classif.idlist(confronto) = avv2;
%               end
%            end
%         end

        if mostraVid
        xx = string((tr(k).list));
        video{k}.String = char( strjoin(xx,' -> '));
        end
      end
      tr(k).ko = getTime(tr(k).idlist)>frames-finestra;
    end

    if mostraVid
    title(['iter:' num2str(j)])
```

```
    drawnow %limitrate
    end
end
```

Terminate le iterazioni si ritiene di aver fatto una ricerca sufficientemente approfondita e ogni filone viene incrementato concatenando la sequenza più lunga mai registrata dal traveller.

```
for k=1:ntr
    try
    tatoo{k} = cat(2,tatoo{k},tr(k).best);
    trasla(k) = timetravel(tr(k).idbest(end),trasla(k))+5;
    storia(k) = storia(k)+"||"+strjoin(tr(k).idbest,"_");
    catch
       disp(['ho avuto problemi con il traveller: ' num2str(k)])
    end

    % se un target supera i 105 debris target, continua diventa 0 e il ciclo while si blocca.
    continua = continua * (length(tatoo{k})<debris_target);
end

% se tutti i debris sono KO allora continua diventa 0
continua = continua * (~all(cat(2,tr(:).ko)));

%[classif.time,trasla] = nuovo(classif.idlist,trasla);
%save(['klassifica' num2str(i) '.mat'], 'classif')
disp(['Ho finito la sequenza ' num2str(i)])

i = i+1;
%clear classif
end
```

Salvo il risultato:

```
save(titolo,'tatoo','storia','DV')
case 0
    load tatoo2.mat
end
```

Funzioni ausiliarie

Ecco lo schema con cui ogni sequenza viene incrementata:

- cerco nodi consecutivi all'ultimo nodo della lista con la funzione *successors*
- se ci sono successori: rimuovo tutti i detriti che sono stati collezionati dal filone e dalla lista stessa con la funzione *setdfiff*
- se l'insieme non è vuoto anche dopo l'applicazione di questo filtro: salvo l'insieme memorizzandolo in una delle caratteristiche del traveller (*pox*) e di questo insieme scelgo casualmente un nodo per ampliare la lista
- si ripete la procedura.

```
function [tr,extra] = esplora(g,tr,varargin)
s = successors(g,tr.idlist(end));
vuoto = false;
extra = [];

if ~isempty(s)
    % il nodo possiede successori
    sn = getNode(s);
    if ~isempty(varargin)
    extra = varargin{1};
    end
    % rimuovo quelli già raccolti in precedenza
    [sn,ia] = setdiff(sn,[tr.list extra]);



    if ~isempty(sn)
    % ne sono avanzati di nuovi, scelgo casualmente
    id = randi(length(sn));
    % incremento la sequenza
    tr.pox{1,end+1} = s(ia);
    tr.idlist(end+1) = tr.pox{end}(id);
    tr.list(end+1) = sn(id);
    else
    vuoto = true;
    end
else
    vuoto = true;
end
```

Se l'ultimo nodo della lista non può indicare nessun successore (sia perché non ne possiede sia perché ne possiede alcuni ma sono stati già presi) si andrà incontro ad una modifica della lista anziché ad un suo ampliamento. Questa procedura è descritta nella funzione *trancia*.

```
if vuoto
    % nodo sterile o incompatibile, lo rimuovo
    [tr,extra]=trancia(tr,extra);
end
```

La funzione *trancia* ha come scopo primario quello di accedere all'insieme dei nodi alternativi correlato ad ogni nodo di cui si compone la lista e in mancanza di una successione sostituire il nodo nella lista con uno delle alternative. E' importante che questo processo sia ben strutturato per evitare comportamenti ridondanti e si deve tenere conto che in alcuni casi si devono effettuare sostituzioni sempre più in profondità con un uso ricorsivo della funzione.

```
    function [tr,extra]=trancia(tr,extra)

        if ~isempty(tr.pox)
        % elimino l'ultimo nodo dall'insieme dei nodi alternativi
```

```
        tr.pox{end}(tr.pox{end}==tr.idlist(end)) = [];
        % lo rimuovo anche dalla lista
        tr.list(end) = [];
        tr.idlist(end) = [];

        % se l'insieme non è vuoto, metto un altro nodo
        if ~isempty(tr.pox{end})
        % scelto casualmente
        idx = randi(length(tr.pox{end}));
        % incremento la lista
        tr.idlist(end+1) = tr.pox{end}(idx);
        tr.list(end+1) =getNode(tr.idlist(end));
        else
        % se invece l'insieme delle alternative è vuoto, si rimuove un altro nodo dalla
        % sequenza:
        %0) elimino l'ultimo insieme delle alternative poichè vuoto e
        %quindi inutilizzabile
        tr.pox(end) = [];
        % il penultimo (che ora è già ultimo) nodo è diventato sterile, va rimosso.
        %1) reitero i processi prima svolti:
        tr = trancia(tr,extra);
        end
        else % se invece ho rimosso tutto (può succedere quando il filone si avvicina ai 105 de
            % mi trovo in una situazione critica, poichè devo
            % reinizializzare il traveller con un nuovo nodo di partenza.
        if length(tr.best)>2
        extra = cat(2,extra,tr.best);

        tr.idbest = [];
        end
        tr=nuovaPiscina(tr, extra);
        end
    end % end della funzione trancia

end % end della funzione esplora

function s = getNode(str)
    s=uint8(double(string(str)));
end

function s=getTime(str)
    s = uint16(rem(double(string(str)),1)*1000);
end

function s=getId(n,t)
    s=(n+t/1000)+"";
end

% function [time,trasla] = nuovo(idlist,trasla)
%     massimo = 0;
%     classif.L = size(idlist,1);
%         for j=1:classif.L
%         aa = char(strrep(idlist(j),'_',' '));
%         ultimo = strfind(aa,'.');
```

```
%         marker = str2double(aa(ultimo(end)+1:end));
%         massimo = max(massimo,marker);
%         trasla = marker;
%         end
%     time = [trasla massimo];
%     trasla = trasla+massimo;
% end

function trasla=timetravel(stringa,~)
    aa = char(strrep(stringa,'_',' '));
    ultimo = strfind(aa,'.');
    marker = str2double(['0.' aa(ultimo(end)+1:end)])*1000;
    trasla = marker;
end


function tr=nuovaPiscina(tr, tatoo)
    global finestra tab
    tr.mover = tr.mover+finestra;
        if tr.mover<size(tab,2)-finestra
        [starter,timess] = find(tab(:,(1:finestra)+tr.mover));
        [new_starter,ia]=setdiff(starter,tatoo);

            if ~isempty(new_starter)
            new_timess = timess(ia);
            piscina = getId(new_starter,new_timess+tr.mover);
            id = randperm(length(new_starter),1);

            tr.pox = cell(1,0);
            tr.pox{1} = piscina;

            tr.idlist(1) = tr.pox{1}(id);
            tr.list = getNode(tr.idlist(1));
            tr.nbest = 0;

            else
             %tr.mover = tr.mover+finestra;
                 if tr.mover<size(tab,2)-finestra
                 tr=nuovaPiscina(tr, tatoo);
                 else
                 tr.ko = true;
                 end
            end
        end
end
```