antisim

```
function S = antisim(v)

% costruisce la matrice antisimmetrica S (3x3) a partire dal vettore v
% Uso: S = antisim(v)

S=[0 -v(3) v(2); v(3) 0 -v(1); -v(2) v(1) 0];
```

cross

```
function c = cross(v,x)

% Uso: c = cross(v,x)
%
% fornisce il prodotto vettoriale di due vettori colonna (3x1) v e x
% usa la matrice antisimmetrica S
% l'uscita e` un vettore colonna c

v=v(:); x=x(:);
S=antisim(v);
c=S*x;
```

crossprod

```
function c = crossprod(v,x)

% Uso: c = crossprod(v,x)
%
% fornisce il prodotto vettoriale di due vettori colonna (3x1) v e x
% usa la matrice antisimmetrica S
% l'uscita e` un vettore colonna c

v=v(:); x=x(:);
S=antisim(v);
c=S*x;
```

denhart

```
function T = denhart(d,teta,a,alfa)

% Uso:  T = denhart(d,teta,a,alfa)
%
% Costruisce la matrice omogenea T (4x4) associata alla
% trasformazione da un riferimento a quello contiguo
% secondo le convenzioni di Denavit-Hartenberg
% con parametri teta, alfa, a, d
% Nota: gli angoli sono in gradi

T = eye(4);

calfa = cosd(alfa);
salfa = sind(alfa);
cteta = cosd(teta);
steta = sind(teta);

T(1,1) = cteta;
T(1,2) = steta;
T(1,4) =    -a;

T(2,1) = -calfa*steta;
T(2,2) =  calfa*cteta;
T(2,3) =        salfa;
T(2,4) =     -d*salfa;

T(3,1) =  salfa*steta;
T(3,2) = -salfa*cteta;
T(3,3) =        calfa;
T(3,4) =     -d*calfa;
```

eul2rot

```
function R = eul2rot (Angles)

% Uso:  R = eul2rot(Angles)
%
% Calcola la matrice di rotazione R
% a partire dagli angoli di eulero espressi in gradi
% Angles(1) = Phi, Angles(2) = Theta, Angles(3) = Psi

F  = grad2rad(Angles(1));
T  = grad2rad(Angles(2));;
P  = grad2rad(Angles(3));;

cf = cos(F);
ct = cos(T);
cp = cos(P);

sf = sin(F);
st = sin(T);
sp = sin(P);

R(3,3) =  ct;
R(1,1) =  cf * cp - sf * ct * sp;
R(1,2) = -cf * sp - sf * ct * cp;
R(1,3) =  sf * st;
R(2,1) =  sf * cp + cf * ct * sp;
R(2,2) = -sf * sp + cf * ct * cp;
R(2,3) = -cf * st;
R(3,1) =  st * sp;
R(3,2) =  st * cp;
```

eulero

```
function [s,h] = eulero(u,teta)

% Uso [s,h] = eulero(u,teta)
%
% calcola i parametri di Eulero s e il quaternione h
% a partire da asse u e angolo di rotazione teta espresso in gradi

a=grad2rad(teta);

s=[u(1)*sin(a/2) u(2)*sin(a/2) u(3)*sin(a/2) cos(a/2)]';

h(1)=s(4);
h(2)=s(1);
h(3)=s(2);
h(4)=s(3);
```

gms2grad

```
function gradi = gms2grad(gms)
%
% converte i gradi, minuti, secondi in gradi decimali
% gms(1) = gradi
% gms(2) = minuti
% gms(3) = secondi

gradi=gms(1)+gms(2)/60+gms(3)/3600;
```

grad2gms

```
function gms = grad2gms (g)
%
% converte i gradi decimali in gradi, minuti, secondi
%
% B Bona, DAUIN, POLITO

gradi_dec=g;
gms(1)=fix(gradi_dec);
min_dec=(gradi_dec-gms(1))*60;
gms(2)=fix(min_dec);
gms(3)=(min_dec-gms(2))*60;
```

grad2rad

```
function r = grad2rad (g)

% Uso: r = grad2rad (g)
%
% Calcola il valore espresso in radianti r
% dell'angolo g espresso in gradi

% inutile se si usano le funzioni sind e cosd che accettano angoli in gradi
%
% B Bona, DAUIN, POLITO

r = pi*g/180;
```

isrot

```
function x = isrot (R)

% Uso: isrot(R)
% verifica che la matrice R sia una matrice di rotazione
% fissa lo zero numerico pari alla variabile zeroeps
%      isrot(R)==0 matrice di rotazione
%      isrot(R)~=0 errore
%
% B Bona, DAUIN, POLITO

zeroeps=5E-6;
stringp=num2str(zeroeps);
warning=strcat('Matrix is not orthonormal, i.e. norm(R''R-I) > ',stringp);

[r c] = size(R);
x = 1;
if r ~= 3
   disp ('Matrix has not 3 rows')
elseif c ~= 3
    disp ('Matrix has not 3 columns')
elseif norm ( R * R' - eye(3) ) > zeroeps
    warning
elseif det (R) < 0
    disp ('Matrix determinant is -1')
else x = 0;
end
```

matqprod1

```
function F = matqprod1(h)

% calcola la matrice F(h) che serve ad effettuare il prodotto tra quaternioni
% h*g = F(h)*g
%
% B Bona, DAUIN, POLITO

F=[h(1) -h(2) -h(3) -h(4) ; ...
   h(2)  h(1) -h(4)  h(3) ; ...
   h(3)  h(4)  h(1) -h(2) ; ...
   h(4) -h(3)  h(2)  h(1) ];
```

matqprod2

```
function F = matqprod2(g)

% calcola la matrice F(g) che serve ad effettuare il prodotto tra quaternioni
% h*g = F(g)*h
%
% B Bona, DAUIN, POLITO

F=[g(1) -g(2) -g(3) -g(4) ; ...
   g(2)  g(1)  g(4) -g(3) ; ...
   g(3) -g(4)  g(1)  g(2) ; ...
   g(4)  g(3) -g(2)  g(1) ];
```

peu2rot

```
function R = peu2rot(s)

% Uso:  R = Ppeu2rot(s)
%
% costruisce la matrice di rotazione R
% a partire dal vettore s dei parametri di Eulero
%
% B Bona, DAUIN, POLITO

R(1,1)=s(1)^2-s(2)^2-s(3)^2+s(4)^2;
R(1,2)=2*(s(1)*s(2)-s(3)*s(4));
R(1,3)=2*(s(1)*s(3)+s(2)*s(4));

R(2,1)=2*(s(1)*s(2)+s(3)*s(4));
R(2,2)=-s(1)^2+s(2)^2-s(3)^2+s(4)^2;
R(2,3)=2*(s(2)*s(3)-s(1)*s(4));

R(3,1)=2*(s(1)*s(3)-s(2)*s(4));
R(3,2)=2*(s(2)*s(3)+s(1)*s(4));
R(3,3)=-s(1)^2-s(2)^2+s(3)^2+s(4)^2;
```

peul2tom

```
function T = peul2tom(p)

% Uso:  T = peul2tom(p)
%
% Calcola la matrice di trasformazione omogenea T
% a partire dal vettore di coordinate omogenee p (6x1)
% con gli angoli espressi come angoli di Eulero in gradi
%
% B Bona, DAUIN, POLITO


    d(1) = p(1);
    d(2) = p(2);
    d(3) = p(3);

    R = eul2rot([p(4) p(5) p(6)]);

    T = rd2tom(R,d);
```

pianifrot

```
function [R1,R2,R3] = pianifrot(R0,Rf,s)

% Uso: [R1,R2,R3] = pianifrot(alfa0,alfaf,s)
%
% Pianificazione di una matrice di rotazione usando i tre metodi:
%
% calcolo di R(s) per 0<=s<=1 assegnato
% a partire dalla matrice iniziale R0
% per giungere alla matrice finale Rf
%
% restituisce R1 usando il metodo asse, angolo
%             R2 usando il metodo di scivolamento piano
%             R3 usando il metodo degli angoli di eulero
%
% B Bona, DAUIN, POLITO

if s>=0 & s<=1

% Metodo 1: asse, angolo

   RL=R0'*Rf;
   [u,deltalfa1]=rot2uth(RL);
   R1s=uth2rot(u,s*deltalfa1);
   R1=R0*R1s;

% Metodo 2: scivolamento piano

   k0=R0(:,3);
   kf=Rf(:,3);
   j0=R0(:,2);
   jf=Rf(:,2);
   u2=cross(k0,kf);
   normu2=norm(u2);

   if normu2==0
      RL=R0'*Rf;
      [u,deltalfa2]=rot2uth(RL);
      R2s=uth2rot(u,s*deltalfa2);
      R2=R0*R1s;
   else
      deltabeta=asind(normu2);
      u2=u2/normu2;
      Rubeta=uth2rot(u2,deltabeta);
      jtilde=Rubeta'*j0;
      deltalfa2=asind(norm(cross(jtilde,jf)));
      R2=uth2rot(u2,s*deltabeta)*Rotz(s*deltalfa2);
      R2=R0*R2;
   end

% Metodo 3: angoli di eulero

   alfa0=rot2eul(R0);
   alfaf=rot2eul(Rf);
   deltalfa3=alfaf-alfa0;
   alfas=alfa0 + s*deltalfa3;
   R3=eul2rot(alfas);

else
   disp ('planning variable s is not between 0 and 1')
```

```
    R1='error';R2='error';R3='error';
end
```

proquatA

```
function hg = proquatA(h,g)

% alternativa A al prodotto di quaternioni
% Uso: hg = proquatA(h,g)
% esegue il prodotto hg fra due quaternioni h e g,
% definiti in riga come h=[hr, hv'] con hv vettore colonna (3x1)
%                       g=[gr, gv'] con gv vettore colonna (3x1)
%
% B Bona, DAUIN, POLITO

F=qprodmat1(h);
hg=F*g(:);
```

proquatB

```
function [hg,F] = proquatB(h,g)

% alternativa B al prodotto di quaternioni
% Uso: hg = proquatB(h,g)
% esegue il prodotto hg fra due quaternioni h e g,
% definiti in riga come h=[hr, hv'] con hv vettore colonna (3x1)
%                       g=[gr, gv'] con gv vettore colonna (3x1)
%
% B Bona, DAUIN, POLITO

F=qprodmat2(g);
hg=F*h(:);
```

proquatvet

```
function hx = proquatvet(h,x)

% Uso: hx = proquatvet(h,x)
% esegue il prodotto hx fra un quaternione h e un vettore x,
%  h=[hr, hv'] in riga, con hv vettore colonna (3x1)
%  x vettore colonna (3x1)
%
% B Bona, DAUIN, POLITO

h_con=quatcon(h); % calcolo del coniugato di h
hx=[0 x'];
h1=proquat(h,hx);
hx=proquat(h1,h_con);
```

prorodri

```
function x = prorodri(ra,rb)

% Uso: x = prorodri(ra,rb)
% esegue il prodotto x fra due vettori di Rodrigues ra rb (in colonna)
% per la composizione di due rotazioni
%
% B Bona, DAUIN, POLITO

rv = cross(rb,ra);

x = (ra + rb - rv)/(1 - ra'*rb);
```

proseu

```
function x = proseu(sa,sb)

% Uso: x = proseu(sa,sb)
%
% esegue il "prodotto" fra due vettori sa sb
% di parametri di Eulero (in colonna) per determinare la rotazione composta
%
% B Bona, DAUIN, POLITO

F(1,1) =  sa(4);
F(1,2) = -sa(3);
F(1,3) =  sa(2);
F(1,4) =  sa(1);

F(2,1) =  sa(3);
F(2,2) =  sa(4);
F(2,3) = -sa(1);
F(2,4) =  sa(2);

F(3,1) = -sa(2);
F(3,2) =  sa(1);
F(3,3) =  sa(4);
F(3,4) =  sa(3);

F(4,1) = -sa(1);
F(4,2) = -sa(2);
F(4,3) = -sa(3);
F(4,4) =  sa(4);

x = F * sb;
```

prpy2tom

```
function T = prpy2tom(p)

% Uso:  T = prpy2tom(p)
%
% Calcola la matrice di trasformazione omogenea T
% a partire dal vettore di coordinate omogenee p (6x1)
% con gli angoli espressi come angoli di RPY in gradi
%
% B Bona, DAUIN, POLITO

    d(1) = p(1);
    d(2) = p(2);
    d(3) = p(3);

    R = rpy2rot([p(4) p(5) p(6)]);

    T = rd2tom(R,d);
```

quat2rot

```
function R = quat2rot(h)

% Uso: R = quat2rot(h)
%
% costruisce la matrice di rotazione R
% a partire dal corrispondente quaternione h
%
% B Bona, DAUIN, POLITO

s(4)=h(1);
s(1)=h(2);
s(2)=h(3);
s(3)=h(4);

R(1,1)=s(1)^2-s(2)^2-s(3)^2+s(4)^2;
R(1,2)=2*(s(1)*s(2)-s(3)*s(4));
R(1,3)=2*(s(1)*s(3)+s(2)*s(4));

R(2,1)=2*(s(1)*s(2)+s(3)*s(4));
R(2,2)=-s(1)^2+s(2)^2-s(3)^2+s(4)^2;
R(2,3)=2*(s(2)*s(3)-s(1)*s(4));

R(3,1)=2*(s(1)*s(3)-s(2)*s(4));
R(3,2)=2*(s(2)*s(3)+s(1)*s(4));
R(3,3)=-s(1)^2-s(2)^2+s(3)^2+s(4)^2;
```

quat2rpy

```
function x = quat2rpy(h)

% Uso: x = quat2rpy(h)
%
% costruisce gli angoli RPY [teta_x teta_y teta_z] a partire dal
% corrispondente quaternione h
% la formula utilizzata si trova in
% "Standards for Representation in Autonomous Intelligent Systems" by DRDC
%
% gli angoli sono espressi in gradi
%
% B Bona, DAUIN, POLITO

qs=h(1);
qx=h(2);
qy=h(3);
qz=h(4);

x(3)=atan2((2*(qx*qy+qs*qz)),(2*(qs*qs+qx*qx)-1));

x(2)=asin(2*(qs*qy-qx*qz));

x(1)=atan2((2*(qy*qz+qs*qx)),(2*(qs*qs+qz*qz)-1));

x=rad2grad(x);
```

quatcon

```
function qc = quatcon(q)

% Uso: trasforma il quaternione q nel suo coniugato qc
%
% B Bona, DAUIN, POLITO

qc(:,1)=q(:,1); % BB parte reale al primo posto
qc(:,2)=-q(:,2);
qc(:,3)=-q(:,3);
qc(:,4)=-q(:,4);
```

quatvecrot2

```
function uvu = quatvecrot2(u,v)

% Uso: uvu = quatvecrot2(u,v)
%
% esegue la rotazione del vettore v (in forma quaternionica)
% usando il quaternione unitario u (in forma quaternionica)
% facendo il prodotto tra i quaternioni u^* v u
% il risultato è in forma quaternionica
%
% B Bona, DAUIN, POLITO

ucon=quatcon(u);
a=proquat(v,u);
b=proquat(ucon,a);

uvur=b(1);
uvuv=[b(2) b(3) b(4)]';

uvu =[uvur uvuv'];
```

quatvecrot

```
function uvu = quatvecrot(u,v)

% Uso: uvu = quatvecrot(u,v)
%
% esegue la rotazione del vettore v (in forma quaternionica)
% usando il quaternione unitario u (in forma quaternionica)
% facendo il prodotto tra i quaternioni u v u^*
% il risultato è in forma quaternionica
%
% B Bona, DAUIN, POLITO

ucon=quatcon(u);
a=proquat(v,ucon);
b=proquat(u,a);

uvur=b(1);
uvuv=[b(2) b(3) b(4)]';

uvu =[uvur uvuv'];
```

rad2grad

```
function g = rad2grad (r)

% Uso: g = rad2grad (r)
%
% Calcola il valore in gradi g
% dell'angolo r espresso in radianti
%
% B Bona, DAUIN, POLITO

g = r*180/pi;
```

randrot

```
function R = randrot

% Uso:  R = randrot
%
% Calcola una matrice di rotazione random
%
% B Bona, DAUIN, POLITO

a=-1;b=1;
u=a + (b-a).*rand(3,1);
unorm=norm(u);
un=u/unorm
teta=a + (b-a).*rand(1,1)*180

R=uth2rot(u,teta);
```

rd2tom

```
function T = rd2tom(R,d)

% Uso:
% T = rd2tom(R,d)
% calcola la matrice di trasformazione omogenea T=T_d*T_R
% a partire da matrice di rotazione R (3x3) e vettore traslazione d (3x1)
% T = rd2tom(R)
% calcola la matrice di trasformazione omogenea T=T_R
% a partire da matrice di rotazione R (3x3), assumendo vettore traslazione d=0
% T = rd2tom(d)
% calcola la matrice di trasformazione omogenea T=T_d
% a partire dal vettore di traslazione d, assumendo la matrice di rotazione R=I
%
% B Bona, DAUIN, POLITO

if nargin == 1
   if size(R,1)==size(R,2)
      R=R;
      d=zeros(3,1);
   elseif size(R,2)==1
      d=R;
      R=eye(3);
   else
      disp ('Error in input arguments')
      x='ERROR';
   end
end

if isrot(R) == 0

    T(1,1) = R(1,1);
    T(1,2) = R(1,2);
    T(1,3) = R(1,3);
    T(1,4) = d(1);

    T(2,1) = R(2,1);
    T(2,2) = R(2,2);
    T(2,3) = R(2,3);
    T(2,4) = d(2);

    T(3,1) = R(3,1);
    T(3,2) = R(3,2);
    T(3,3) = R(3,3);
    T(3,4) = d(3);

    T(4,1) = 0;
    T(4,2) = 0;
    T(4,3) = 0;
    T(4,4) = 1;

else

   disp ('Error in input matrix')
   T='ERROR';

end
```

rod2rot

```
function R = rod2rot(r)

% Uso: R = rod2rot(r)
%
% costruisce la matrice di rotazione R
% a partire dal vettore di Rodrigues r
%
% B Bona, DAUIN, POLITO

s(1)=r(1);
s(2)=r(2);
s(3)=r(3);
s(4)=   1;

R(1,1)=s(1)^2-s(2)^2-s(3)^2+s(4)^2;
R(1,2)=2*(s(1)*s(2)-s(3)*s(4));
R(1,3)=2*(s(1)*s(3)+s(2)*s(4));

R(2,1)=2*(s(1)*s(2)+s(3)*s(4));
R(2,2)=-s(1)^2+s(2)^2-s(3)^2+s(4)^2;
R(2,3)=2*(s(2)*s(3)-s(1)*s(4));

R(3,1)=2*(s(1)*s(3)-s(2)*s(4));
R(3,2)=2*(s(2)*s(3)+s(1)*s(4));
R(3,3)=-s(1)^2-s(2)^2+s(3)^2+s(4)^2;

R=1/(1+r(1)^2+r(2)^2+r(3)^2) * R;
```

rot2eul

```
function x = rot2eul (R)

% Uso:  x = rot2eul(R)
%
% Calcola gli angoli di eulero a partire dalla matrice di rotazione R
% x(1) = Phi, x(2) = Theta, x(3) = Psi
% gli angoli sono espressi in gradi
%
% B Bona, DAUIN, POLITO

if isrot(R) == 0

   x(1) = atan2(R(1,3),-R(2,3));
   cf   = cos(x(1));
   sf   = sin(x(1));
   x(2) = atan2(sf*R(1,3)-cf*R(2,3),R(3,3));
   x(3) = atan2(-cf*R(1,2)-sf*R(2,2),cf*R(1,1)+sf*R(2,1));
   x    = rad2grad(x);

else

   disp ('Error in input matrix')
   x='ERROR';

end
```

rot2omeg

```
function [w,tetap] = rot2omeg(R,Rp)

% Uso: [w,tetap] = rot2omeg(R,Rp)
%
% costruisce la velocita` angolare w (vettore colonna unitario 3x1)
% e il modulo di w (tetap)
% a partire dalla matrice di rotazione R e dalla matrice derivata temporale Rp
%
% B Bona, DAUIN, POLITO

if isrot(R) == 0
    w = 0.5.*(cross(R(:,1),Rp(:,1)) + cross(R(:,2),Rp(:,2)) +
cross(R(:,3),Rp(:,3)));
    tetap=norm(w);
    if abs(tetap) > eps
        w = w/tetap;
    end
else
    disp ('Error in input matrix')
    x='ERROR';
end
```

rot2peu

```
function x = rot2peu (R)

% Uso:  s = rot2peu(R)
%
% Calcola i parametri di Eulero s a partire dalla matrice di rotazione R
%
% B Bona, DAUIN, POLITO

if isrot(R) == 0
    x(4) = .5 * sqrt( 1 + R(1,1) + R(2,2) + R(3,3) );
    if x(4) < 1.0E-8
        % disp ('rotation = 180 degrees')
        x(1) = 1;
        x(2) = 0;
        x(3) = 0;
    elseif x(4) == 1
        % disp ('rotation = 0 degrees')
        x(1) = 0;
    x(2) = 0;
    x(3) = 0;
   else
        x(1) = (R(3,2) - R(2,3))/(4 * x(4));
        x(2) = (R(1,3) - R(3,1))/(4 * x(4));
        x(3) = (R(2,1) - R(1,2))/(4 * x(4));
    end
    x = x';
else
    disp ('Error in input matrix')
    x='ERROR';
end
```

rot2quat

```
function x = rot2quat (R)

% Uso:  Angles = rot2quat(R)
%
% Calcola il quaternione corrispondente a partire dalla matrice di rotazione R
%
% B Bona, DAUIN, POLITO

eps =  1.0e-7; % precisione con cui viene calcolato lo zero

if isrot(R) == 0
    s(4) = .5 * sqrt( 1 + R(1,1) + R(2,2) + R(3,3) );
    if norm(s(4)) <= eps
         disp ('rotation = 180 degrees')
        [u,teta]=rot2uth(R);
        s(1)=u(1);
        s(2)=u(2);
        s(3)=u(3);
    elseif norm(s(4)-1) <= eps
         disp ('rotation = 0 degrees')
        s(1) = 0;
        s(2) = 0;
        s(3) = 0;
    else
        s(1) = (R(3,2) - R(2,3))/(4 * s(4));
        s(2) = (R(1,3) - R(3,1))/(4 * s(4));
        s(3) = (R(2,1) - R(1,2))/(4 * s(4));
    end
    x(1) = s(4);
    x(2) = s(1);
    x(3) = s(2);
    x(4) = s(3);
else
    disp ('Error in input matrix')
    x='ERROR';
end
```

rot2quatCS

```
function x = rot2quatCS (R)

% Uso:  Angles = rot2quatCS(R)
%
% Calcola il quaternione corrispondente a partire dalla matrice di rotazione R
% secondo la formula data nell'articolo Siciliano Chiaverini(SC)

%
% B Bona, DAUIN, POLITO

if isrot(R) == 0

    s(4) = .5 * sqrt( 1 + R(1,1) + R(2,2) + R(3,3) );
    s(1) = 0.5*sign(R(3,2) - R(2,3))*sqrt(+R(1,1) - R(2,2) - R(3,3) + 1);
    s(2) = 0.5*sign(R(1,3) - R(3,1))*sqrt(-R(1,1) + R(2,2) - R(3,3) + 1);
    s(3) = 0.5*sign(R(2,1) - R(1,2))*sqrt(-R(1,1) - R(2,2) + R(3,3) + 1);

    x(1) = s(4);
    x(2) = s(1);
    x(3) = s(2);
    x(4) = s(3);
else
    disp ('Error in input matrix')
    x='ERROR';
end
```

rot2quatSK

```
function x = rot2quatSK(R)

% Uso:  Angles = rot2quatSK(R)
%
% Calcola il quaternione corrispondente a partire dalla matrice di rotazione R
% utilizzando l'algoritmo di Sheppard-Klumpp HAL/S
% descritto a pag. 4 del "Interoffice Memorandum IOM 343-79-1199 Oct. 1979"
% by Breckenridge, NASA
%
% B Bona, DAUIN, POLITO

if isrot(R) == 0
    tr=R(1,1)+R(2,2)+R(3,3);
    m=[R(1,1) R(2,2) R(3,3) tr]';
    [y,i]=max(m);
    M=R';
    switch i
        case 1
            s(1)=0.5*sqrt(1+2*m(1,1)-tr);
            s(2)=0.25*(M(1,2)+M(2,1))/s(1);
            s(3)=0.25*(M(1,3)+M(3,1))/s(1);
            s(4)=0.25*(M(2,3)-M(3,2))/s(1);
        case 2
            s(2)=0.5*sqrt(1+2*m(2,2)-tr);
            s(1)=0.25*(M(1,2)+M(2,1))/s(2);
            s(3)=0.25*(M(2,3)+M(3,2))/s(2);
            s(4)=0.25*(M(3,1)-M(1,3))/s(2);
        case 3
            s(3)=0.5*sqrt(1+2*m(3,3)-tr);
            s(1)=0.25*(M(1,3)+M(3,1))/s(3);
            s(2)=0.25*(M(2,3)+M(3,2))/s(3);
            s(4)=0.25*(M(1,2)-M(2,1))/s(3);
        case 4
            s(4)=0.5*sqrt(1+tr);
            s(1)=0.25*(M(2,3)-M(3,2))/s(4);
            s(2)=0.25*(M(3,1)-M(1,3))/s(4);
            s(3)=0.25*(M(1,2)-M(2,1))/s(4);
        otherwise
            disp ('Error in input matrix')
            x='ERROR';
    end
    x(1) = s(4);
    x(2) = s(1);
    x(3) = s(2);
    x(4) = s(3);
else
    disp ('Error in input matrix')
    x='ERROR';
end
```

rot2rod

```
function r = rot2rod(R)

% Uso: r = rot2rod(R)
%
% costruisce il vettore di Rodrigues r a partire dalla matrice di rotazione R
%
% B Bona, DAUIN, POLITO

if isrot(R) == 0

% passo alla rappresentazione asse u + angolo teta

    [u,teta]=rot2uth(R);

% passo alla rappresentazione parametri di Eulero s + quaternione h

    if abs(teta-180) < eps
        disp('Rodrigues vector is undefined: teta = 180')
        r=NaN
    else
        s=Eulero(u,teta);
        r=[s(1) s(2) s(3)]/s(4);
        r=r';
    end
else
        disp('Error in input matrix')
        r='ERROR'
end
```

rot2rpy

```
function x = rot2rpy2 (R)

% Uso:  x = rot2rpy2(R)
%
% Calcola gli angoli di roll, pitch, yaw a partire dalla matrice di
% rotazione R, con R=Rx*Ry*Rz (quella alternativa)
% x(1) = teta_x, x(2) = teta_y, x(3) = teta_z
% gli angoli vanno espressi in gradi
%
% B Bona, DAUIN, POLITO

if isrot(R) == 0

   x(3) = atan2(-R(1,2),R(1,1));
   cz = cos(x(3));
   sz = sin(x(3));

   sx = R(3,1)*sz+R(3,2)*cz;
   cx = R(2,1)*sz+R(2,2)*cz;
   x(1) = atan2(sx,cx);

   x(2) = atan2( R(1,3), -sx*R(2,3)+cx*R(3,3) );
   x = rad2grad(x);

else

   disp ('Error in input matrix')
   x='ERROR';

end
```

rot2rpy

```
function x = rot2rpy (R)

% Uso:  x = rot2rpy(R)
%
% Calcola gli angoli di roll, pitch, yaw a partire dalla matrice di
% rotazione R, con R=Rz*Ry*Rx (quella convenzionale)
% x(1) = teta_x, x(2) = teta_y, x(3) = teta_z
% gli angoli vanno espressi in gradi
%
% B Bona, DAUIN, POLITO

if isrot(R) == 0

   x(1) = atan2(R(3,2),R(3,3));
   cx = cos(x(1));
   sx = sin(x(1));
   x(3) = atan2( -cx*R(1,2)+sx*R(1,3), cx*R(2,2)-sx*R(2,3) );
   x(2) = atan2( -R(3,1), sx*R(3,2)+cx*R(3,3) );
   x = rad2grad(x);

else

   disp ('Error in input matrix')
   x='ERROR';

end
```

rot2uth

```
function [u,teta] = rot2uth (R)

% Uso: [u,teta] = rot2uth(R)
%
% Calcola l'asse u (vettore 3x1 colonna)
% e l'angolo di rotazione teta a partire dalla matrice R
% l'angolo è espresso in gradi
%
% B Bona, DAUIN, POLITO

if isrot(R) == 0
    I = eye(3);
    if abs(norm(R-R')) < 1.0e-5 % rotazione di 180 gradi
        teta = 180;
        M = 0.5*(R+I);
        us(1) = sqrt(M(1,1));
        us(2) = sqrt(M(2,2));
        us(3) = sqrt(M(3,3));
        if abs(us(1)) > 1.0e-6
            u(1) = us(1);
            u(2) = M(1,2)/us(1);
            u(3) = M(1,3)/us(1);
        elseif abs(us(2)) > 1.0e-6
            u(2) = us(2);
            u(1) = M(2,1)/us(2);
            u(3) = M(2,3)/us(2);
        elseif abs(us(3)) > 1.0e-6
            u(3) = us(3);
            u(1) = M(3,1)/us(3);
            u(2) = M(3,2)/us(3);
        end
    elseif abs(norm(R-I)) < 1.0e-5 % rotazione di 0 gradi; il vettore u e` messo
pari a [1 0 0]
        teta = 0;
        u(1) = 1;
        u(2) = 0;
        u(3) = 0;
    else
        s(4) = .5 * sqrt(1 + trace(R));
        s(1) = (R(3,2) - R(2,3))/(4 * s(4));
        s(2) = (R(1,3) - R(3,1))/(4 * s(4));
        s(3) = (R(2,1) - R(1,2))/(4 * s(4));
        amez = acos(s(4));
        a = 2*amez;
        teta = rad2grad(a);
        sa=sin(amez);
        u(1) = s(1)/sa;
        u(2) = s(2)/sa;
        u(3) = s(3)/sa;
    end
    u = u';
    nu=norm(u);
    u=u/nu;

else
    disp ('Error in input matrix')
    x='ERROR';
end
```

rotx

```
function x = rotx(a)

% Uso:  R = rotx(a)
%
% Costruisce la matrice R di rotazione elementare
% intorno all'asse x di un angolo a espresso in gradi
%
% B Bona, DAUIN, POLITO

x = eye(3);

x(2,2) = cosd(a);
x(3,3) = x(2,2);
x(3,2) = sind(a);
x(2,3) = -x(3,2);
```

roty

```
function x = roty(b)

% Uso:  R = roty(b)
%
% Costruisce la matrice R di rotazione elementare
% intorno all'asse y di un angolo b espresso in gradi
%
% B Bona, DAUIN, POLITO

x = eye(3);

x(1,1) = cosd(b);
x(3,3) = x(1,1);
x(1,3) = sind(b);
x(3,1) = -x(1,3);
```

rotz

```
function x = rotz(c)

% Uso:  R = rotz(c)
%
% Costruisce la matrice R di rotazione elementare
% intorno all'asse z di un angolo c espresso in gradi
%
% B Bona, DAUIN, POLITO

x = eye(3);

x(2,2) = cosd(c);
x(1,1) = x(2,2);
x(2,1) = sind(c);
x(1,2) = -x(2,1);
```

rpunto

```
function Rp = rpunto(w,tetap,R)

% Uso: Rp = rpunto(w,tetap,R)
%
% costruisce matrice derivata temporale Rp
% a partire dalla velocita` angolare w (vettore unitario 3x1),
% dalla norma (tetap) di w
% e dalla matrice di rotazione R
%
% B Bona, DAUIN, POLITO

if isrot(R) == 0
    Rp = antisim(w)*R*tetap;
else
    disp ('Error in input matrix')
    x='ERROR';
end
```

rpy2rot

```
function R = rpy2rot(x)

% Uso:  R = rpy2rot(x)
%
% Calcola la matrice di rotazione R a partire dagli angoli di Roll, Pitch, Yaw
% x(1) = teta_x = Roll,
% x(2) = teta_y = Pitch
% x(3) = teta_z = Yaw
% gli angoli sono espressi in gradi
%
% B Bona, DAUIN, POLITO

cx = cosd(x(1));
cy = cosd(x(2));
cz = cosd(x(3));

sx = sind(x(1));
sy = sind(x(2));
sz = sind(x(3));

R(1,1) =  cz*cy;
R(1,2) =  cz*sy*sx-sz*cx;
R(1,3) =  cz*sy*cx+sz*sx;

R(2,1) =  sz*cy;
R(2,2) =  sz*sy*sx+cz*cx;
R(2,3) =  sz*sy*cx-cz*sx;

R(3,1) =  -sy;
R(3,2) =  cy*sx;
R(3,3) =  cy*cx;
```

tom

```
function y = tom(T,x)

% Uso:
% y = tom(T,x)
% Calcola la trasformazione del vettore x (3x1) secondo la matrice omogenea (4x4)
T
% e restituisce il vettore trasformato y (3x1)
%
% B Bona, DAUIN, POLITO

if size(T,1)~=4 | size(T,2)~=4
   disp ('Error in input homogeneous matrix ')
   y='error';
elseif isrot(T(1:3,1:3))==1
   disp ('Error in input rotation matrix ')
   y='error';
end
xh=[x(:); 1];
yh=T*xh;
y=yh(1:3);
```

tom2peul

```
% Uso:  p = tom2peul(T)
%
% Calcola il vettore di coordinate omogenee p (6x1)
% con gli angoli espressi come angoli di Eulero in gradi
% a partire dalla matrice di trasformazione omogenea T
%
% B Bona, DAUIN, POLITO

[R,d] = tom2rd(T);

p(1) = d(1);
p(2) = d(2);
p(3) = d(3);

alfa = rot2eul(R);

p(4) = alfa(1);
p(5) = alfa(2);
p(6) = alfa(3);

p=p(:);
```

tom2prpy

```
function p = tom2prpy(T)

% Uso:  p = tom2prpy(T)
%
% Calcola il vettore di coordinate omogenee p (6x1)
% con gli angoli espressi come angoli di RPY in gradi
% a partire dalla matrice di trasformazione omogenea T
%
% B Bona, DAUIN, POLITO

[R,d] = tom2rd(T);

p(1) = d(1);
p(2) = d(2);
p(3) = d(3);

alfa = rot2rpy(R);

p(4) = alfa(1);
p(5) = alfa(2);
p(6) = alfa(3);

p=p(:);
```

tom2rd

```
function [R,d] = tom2rd(T)

% Uso:
% [R,d] = tom2rd(T)
% Calcola la matrice di rotazione R (3x3) e il vettore di traslazione d (3x1)
% a partire dalla matrice di trasformazione omogenea (4x4) T
%
% B Bona, DAUIN, POLITO

    R(1,1) = T(1,1);
    R(1,2) = T(1,2);
    R(1,3) = T(1,3);
      d(1) = T(1,4);

    R(2,1) = T(2,1);
    R(2,2) = T(2,2);
    R(2,3) = T(2,3);
      d(2) = T(2,4);

    R(3,1) = T(3,1);
    R(3,2) = T(3,2);
    R(3,3) = T(3,3);
      d(3) = T(3,4);

    d=d(:);

if isrot(R) ~= 0

   disp ('Error in input matrix, R has been set to eye(3), d to zero')
   R=eye(3);
   d=[0 0 0]'

end
```

tominv

```
function Tinv = tominv(T)

% Uso:  Tinv = tominv(T)
%
% Calcola l'inversa della matrice di trasformazione omogenea T
%
% B Bona, DAUIN, POLITO

[R,d] = tom2rd(T);
Tinv = rd2tom(R',-R'*d);
```

u2omega

```
function [w,tetap1] = u2omega(u,up,teta,tetap)

% Uso: [w,tetap1] = u2omega(u,up,teta,tetap)
%
% costruisce velocita` angolare w normalizzata e la sua norma tetap
% a partire da:
% asse u vettore (3x1) anche non normalizzato, derivata asse up vettore (3x1),
% angolo teta (in gradi), derivata tetap (in rad/s)
% NOTA BENE: tetap1 in uscita puo` essere diversa da tetap in ingresso
%            se norm(up) ~= 0
%
% B Bona, DAUIN, POLITO

nu=norm(u);
if nu ~= 0
    u = u/nu;
    nup=norm(up);
    if nup ~= 0
        up = up/nup;
    end
    w = tetap*u + sind(teta)*up + (1-cosd(teta))*(cross(u,up));
    tetap1 = norm(w);
    if abs(tetap1) > eps
        w = w/tetap1;
    end
    w=w';
else
    disp('u norm = 0')
end
```

uth2rot

```
function R = uth2rot(u,teta)

% Uso: R = uth2rot(u,teta)
%
% calcola la matrice R
% a partire da asse u ed angolo di rotazione teta (in gradi)
%
% B Bona, DAUIN, POLITO

S=antisim(u);
t=grad2rad(teta);

n=norm(u);

R = eye(3) + sin(t)/n*S + (1-cos(t))/n^2*S^2;
```

uth2rotCS

```
function R = uth2rotCS(u,teta)

% Uso: R = uth2rotCS(u,teta)
%
% calcola la matrice R
% a partire da asse u ed angolo di rotazione teta (in gradi)
% usando la formula data da Chiaverini e Siciliano
%
% B Bona, DAUIN, POLITO

n=norm(u);
r=u/n;
S=antisim(r);

R = eye(3)*cosd(teta) + (1-cosd(teta))*r*r' - sind(teta)*S;
```