

Beam optics support functions 5D

Volker Ziemann, 220101

In this script we define the functions for the 5D beam optics calculations, such as `calcmat()` that is frequently used in other calculations. All described functions reside in the subdirectory 5D. Any scripts using these functions need to include that subdirectory with the command `"addpath ./5D"`.

The function `calcmat()` to calculate all transfer matrices

The following function receives the beamline description as input and returns

- `Racc(5,5,nmat)`: transfer matrices from the start to the end of each segment, such that `R(:, :, end)` is the transfer matrix from the start to the end of the beamline.
- `spos`: position along the beamline after each segment, useful when plotting.
- `nmat`: number of segments
- `nlines`: number of lines in the beamline
- `ibl`: reverse lookup of the line from the segment, such that `line=ibl(k)` with `k=1, ..., nmat`.

```
function [Racc,spos,nmat,nlines,ibl]=calcmat(beamline)
ndim=size(DD(1),1);
nlines=size(beamline,1);           % number of lines in beamline
nmat=sum(beamline(:,2))+1;          % sum over repeat-count in column 2
Racc=zeros(ndim,ndim,nmat);         % matrices from start to element-end
Racc(:, :, 1)=eye(ndim);            % initialize first with unit matrix
ibl=zeros(nmat,1);                  % position in beamline file
spos=zeros(nmat,1);                 % longitudinal position
ic=1;                               % element counter
ibl(1)=1;
for line=1:nlines                   % loop over input elements
    for seg=1:beamline(line,2)      % loop over repeat-count
        ic=ic+1;                    % next element
        Rcurr=eye(ndim);            % matrix in next element
        switch beamline(line,1)
            case 1 % drift
                Rcurr=DD(beamline(line,3));
            case 2 % thin quadrupole
                Rcurr=Q(beamline(line,4));
            case 4 % sector dipole
                phi=beamline(line,4)*pi/180; % convert to radians
                rho=beamline(line,3)/phi;
                Rcurr=SB(beamline(line,3),rho);
            case 5 % thick quadrupole
                Rcurr=QQ(beamline(line,3),beamline(line,4));
            case 20 % coordinate roll
                Rcurr=ROLL(beamline(line,4));
            case 60 % skew quadrupole
                Rcurr=SQ(beamline(line,4));
```

```

case 201 % phase advance in X
    phasex=beamline(line,4)*pi/180;
    cx=cos(phasex); sx=sin(phasex);
    Rcurr(1,1)=cx; Rcurr(1,2)=sx; Rcurr(2,1)=-sx; Rcurr(2,2)=cx;
case 202 % phase advance in Y
    phasey=beamline(line,4)*pi/180;
    cy=cos(phasey); sy=sin(phasey);
    Rcurr(3,3)=cy; Rcurr(3,4)=sy; Rcurr(4,3)=-sy; Rcurr(4,4)=cy;
otherwise
    current_line=line;
    disp('unsupported code')
end
Racc(:, :, ic)=Rcurr*Racc(:, :, ic-1); % concatenate
spos(ic)=spos(ic-1)+beamline(line,3); % position of element
ibl(ic)=line; % reverse lookup of segment
end
end
end

```

Transfer matrix for a drift space $DD(L)$

The function `DD()` receives the length L of a drift space and returns the 5x5 transfer matrix `out` for a drift space.

```

function out=DD(L)
out=eye(5);
out(1,2)=L;
out(3,4)=L;
end

```

Transfer matrix for a thin-lens quadrupole $Q(F)$

The function `Q()` receives the focal length F as input and returns the 5x5 transfer matrix `out` for a thin-lens quadrupole.

```

function out=Q(F)
out=eye(5);
if abs(F)<1e-8, return; end
out(2,1)=-1/F;
out(4,3)=1/F;
end

```

Transfer matrix for a thick quadrupole $QQ(L, k)$

The function `QQ()` receives the length L and $k1$ as input and returns the 5x5 transfer matrix `out` for a thick quadrupole.

```

function out=QQ(L,k)
ksq=sqrt(abs(k));
out=eye(5);
if abs(k) < 1e-6
    out(1,2)=L;
    out(3,4)=L;
end

```

```

else
    A=[cos(ksq*L),sin(ksq*L)/ksq;-ksq*sin(ksq*L),cos(ksq*L)];
    B=[cosh(ksq*L),sinh(ksq*L)/ksq;ksq*sinh(ksq*L),cosh(ksq*L)];
    if k>0
        out(1:2,1:2)=A;
        out(3:4,3:4)=B;
    else
        out(1:2,1:2)=B;
        out(3:4,3:4)=A;
    end
end
end
end

```

Transfer matrix for a thin-lens skew quadrupole SQ(F)

The function SQ() receives the focal length F as input and returns the 5x5 transfer matrix `out` for a thin-lens skew quadrupole.

```

function out=SQ(F)
out=eye(5);
if abs(F)<1e-8, return; end
out(2,3)=1/F;
out(4,1)=1/F;
end

```

Transfer matrix for a sector dipole SB(L, rho)

The function SB() receives the length L and bending radius ρ of a horizontally deflecting sector dipole magnet and returns its 5x5 transfer matrix `out`.

```

function out=SB(L,rho)
phi=L/rho;
out=eye(5);
out(3,4)=L;
if abs(phi)<1e-8
    out(1,2)=L;
else
    out(1:2,1:2)=[cos(phi),rho*sin(phi); ...
                  -sin(phi)/rho,cos(phi)];
    out(1:2,5)=[rho*(1-cos(phi)); ...
                sin(phi)];
end
end

```

Transfer matrix for coordinate rotation ROLL(phi)

The function ROLL() receives the roll angle ϕ (in degree) around the s-direction as input and returns the corresponding 5x5 transfer matrix `out`.

```

function out=ROLL(phi) % phi in degree
c=cos(phi*pi/180); s=sin(phi*pi/180);
out=eye(5);
out(1,1)=c; out(1,3)=s; out(2,2)=c; out(2,4)=s;

```

```
out(3,1)=-s; out(3,3)=c; out(4,2)=-s; out(4,4)=c;
end
```

R2beta()

The function `R2beta()` receives a transfer matrix `R` as input and returns the "tune" $Q = \mu/2\pi$ for the transfer matrix `R`, as well as the periodic Twiss parameters α , β , and γ following Equation 3.60.

```
function [Q,alpha,beta,gamma]=R2beta(R)
mu=acos(0.5*(R(1,1)+R(2,2)));
if (R(1,2)<0), mu=2*pi-mu; end
Q=mu/(2*pi);
beta=R(1,2)/sin(mu);
alpha=(0.5*(R(1,1)-R(2,2)))/sin(mu);
gamma=(1+alpha^2)/beta;
end
```

plot_betas()

The function `plot_betas()` receives the beamline description and the initial 5x5 beam matrix `sigma0` as input and produces a plot of the horizontal and the vertical beta function. This function assumes that the emittance of `sigma0` is 1, or $\det \sigma_0 = 1$ in both planes, such that $\sigma_{11} = \beta_x$ and $\sigma_{33} = \beta_y$ are the beta functions. It then uses Equation 3.43 to propagate σ .

```
function plot_betas(beamline,sigma0)
[Racc,spos]=calcmat(beamline);
betax=zeros(1,length(spos)); betay=betax;
for k=1:length(spos)
    sigma=Racc(:, :,k)*sigma0*Racc(:, :,k)';
    betax(k)=sigma(1,1); betay(k)=sigma(3,3);
end
plot(spos,betax,'k',spos,betay,'r-.','LineWidth',2);
xlabel(' s[m]'); ylabel('\beta_x,\beta_y [m]')
legend('\beta_x','\beta_y')
axis([0, max(spos), 0, 1.05*max([betax,betay])])
end
```

plot_betas_unrolled()

The function `plot_betas_unrolled()` receives the beamline description and the parameters `flipped` and `show` as input and returns arrays with the periodic beta functions `beta1` and `beta2`, the periodic dispersion `disp` and the evolution of the tunes `Qtune` along the beamline. If `flipped==1` it uses the flipped solution from Sagan and Rubin, if `show==1`, it produces nicely annotated plots. Note that this routine is adapted by hand to ensure that the fractional part of the vertical tunes is smaller than that of the horizontal tune. It also unrolls the coordinate system, before returning the beta functions and dispersions.

```
function [beta1,beta2,disp,Qtune]=plot_betas_unrolled(beamline,flipped,show)
if nargin==1, flipped=0; end % default, use SR eq.8+9
if nargin==2, show=1; end
unroll=1; if flipped==-1, unroll=0; end % flipped=-1 turns of unrolling
[Racc,spos,nmat,nlines,ibl]=calcmat(beamline);
Rturn=Racc(:, :,end);
```

```

[O,A,T,p]=sagrub(Rturn,flipped);
if p(4) < p(1), flipped=~flipped; [O,A,T,p]=sagrub(Rturn,flipped); end % ensure [Qy]>[Qx]
S1=A*T; S1inv=inv(S1);
D0=periodic_dispersion(Rturn);
inroll=0; Rs=eye(5); % prepare for the unrolling of coordinate rotations
beta1=zeros(1,nmat); beta2=beta1; disp=zeros(nmat,2);
Qtune=zeros(nmat,2); dQ11=0; dQ21=0; % needed for phase advances
for k=1:nmat
    if unroll==1 && beamline(ibl(k),1)==20 % coordinate rotation found
        if inroll==0 % rotation found and NOT in rolled region
            inroll=1; Rs=ROLL(-beamline(ibl(k),4)); % Rs = unroll matrix
        else % found the un-rotation after the element
            inroll=0; Rs=eye(5); % no unrolling needed any more
        end
    end
    R=Rs*Racc(:, :, k)*Rturn*inv(Rs*Racc(:, :, k)); % move FTM to point k
    [O,A,T,p]=sagrub(R,flipped); beta1(k)=p(3); beta2(k)=p(6); % betas
    D=Rs*Racc(:, :, k)*D0; disp(k,1)=D(1); disp(k,2)=D(3); % dispersions
    if k>1 %.....phase advances from second element onwards
        R12=Rs*Racc(:, :, k); % unrolled!
        S2=A*T;
        OO=S2*R12(1:4,1:4)*S1inv; % two rotations on diagonal, 211224
        if abs(det(OO(1:2,1:2))) < 1e-12 % flipped mode
            [O,A,T,p]=sagrub(R,~flipped); % use the other mode
            S2=A*T;
            OO=S2*R12(1:4,1:4)*S1inv;
        end
        mu1=acos(0.5*(OO(1,1)+OO(2,2))); % phase a
        if OO(1,2)<-1e-12, mu1=2*pi-mu1; end; dQ1=mu1/(2*pi);
        Qtune(k,1)=Qtune(k-1,1)+dQ1-dQ11;
        if dQ11 > dQ1+0.25, Qtune(k,1)=Qtune(k,1)+1; end
        mu2=acos(0.5*(OO(3,3)+OO(4,4))); % phase b
        if OO(3,4)<-1e-12, mu2=2*pi-mu2; end; dQ2=mu2/(2*pi);
        Qtune(k,2)=Qtune(k-1,2)+dQ2-dQ21;
        if dQ21 > dQ2+0.25, Qtune(k,2)=Qtune(k,2)+1; end
        dQ11=dQ1; dQ21=dQ2;
    end
end
if show==0, return; end
%..... only plotting below
subplot(2,1,1);
plot(spos,real(beta1), 'k', spos,real(beta2), 'r-.', 'LineWidth', 2);
xlim([min(spos),max(spos)])
xlabel('s [m]'); ylabel('\beta_a, \beta_b'); legend('\beta_a', '\beta_b')
title(['Beta functions: Q_a= ', num2str(Qtune(end,1), '%8.3f'), ' Q_b= ', num2str(Qtune(end,2), '%8.3f')])
drawmag(beamline,1,2)
set(gca, 'FontSize', 16)

subplot(2,1,2);
plot(spos,real(disp(:,1)), 'k', spos,real(disp(:,2)), 'r-.', 'LineWidth', 2);
xlim([min(spos),max(spos)])
xlabel('s [m]'); ylabel('D_a, D_b'); legend('D_a', 'D_b')
title('Dispersions')
set(gca, 'FontSize', 16)

```

```

set(gcf,'Position',[270,-100,1400,900])

figure
plot(spos,real(Qtune(:,1)),'k',spos,real(Qtune(:,2)),'r-.','LineWidth',2)
xlabel('s [m]'); ylabel('\mu_a/2\pi, \mu_b/2\pi')
legend('\mu_a/2\pi', '\mu_b/2\pi', 'Location', 'NorthWest');
xlim([min(spos),max(spos)]);
set(gca,'FontSize',16)
set(gcf,'Position',[270,-100,1400,900])
end

```

plot_sigmatrix()

The function `plot_sigmatrix()` receives the beamline description and the initial 5x5 beam matrix `sigma0` as input and produces a plot of the horizontal and the vertical beam sizes σ_x and σ_y . It uses Equation 3.43 to propagate σ . Note that this routine always plots the beam sizes in the lab frame. If it detects a coordinate rotation (with element code 20) it always rolls back the coordinate system before plotting σ_x and σ_y .

```

function [sigx,sigy]=plot_sigmatrix(beamline,sigma0)
[Racc,spos,nmat,nlines,ibl]=calcmat(beamline);
inroll=0; Rs=eye(5);
sigx=zeros(1,nmat); sigy=sigx;
for k=1:nmat
    if beamline(ibl(k),1)==20 % coordinate rotation found
        if inroll==0
            inroll=1; Rs=ROLL(-beamline(ibl(k),4));
        else
            inroll=0; Rs=eye(5);
        end
    end
    RR=Rs*Racc(:, :, k); sig=RR*sigma0*RR';
    sigx(k)=sqrt(sig(1,1)); sigy(k)=sqrt(sig(3,3));
end
plot(spos,sigx*1e3,'k',spos,sigy*1e3,'r-.','LineWidth',2)
xlim([min(spos),max(spos)])
xlabel('s [m]'); ylabel('\sigma_x, \sigma_y [mm]')
legend('\sigma_x', '\sigma_y')
title('Beam sizes')
% set(gca,'FontSize',16)
% set(gcf,'Position',[800,-200,1400,500])
end

```

periodic_beammatrix()

The function `periodic_beammatrix()` receives the 5x5 transfer matrix R_{end} and the emittances `epsx` and `epsy` as input and returns the 5x5 beam matrix σ that obeys $\sigma = R_{end}\sigma_{end}^T$. In other words, it is periodic. This routine only works for uncoupled beamlines!

```

function sigma=periodic_beammatrix(Rend,epsx,epsy)
[Qx,alphax,betax,gammax]=R2beta(Rend(1:2,1:2)); % eq. 3.60
[Qy,alphay,betay,gamay]=R2beta(Rend(3:4,3:4));
sigma=zeros(4,4);

```

```
sigma(1:2,1:2)=epsx*[betax,-alphax;-alphax,gammax]; % eq. 3.78
sigma(3:4,3:4)=epsy*[betay,-alphay;-alphay,gammay];
end
```

periodic_beammatrix2()

The function `periodic_beammatrix()` receives the 5x5, possibly coupled, transfer matrix R_{end} and the emittances ϵ_{sx} , ϵ_{sy} and sigp as well as the parameter `flipped` [0,1] as input and returns the 5x5 beam matrix σ that obeys $\sigma = R_{end}\sigma R_{end}^t$. In other words, it is periodic. This routine also works for coupled beamlines! Note that if `flipped==1` the subroutine uses the flipped solution referred to in Sagan and Rubin's PRAB paper.

```
function sigma=periodic_beammatrix2(Rend,epsx,epsy,sigp,flipped)
if nargin<5, flipped=0; end
[O,A,T,para]=sagrub(Rend(1:4,1:4),flipped);
sigma=zeros(5);
iAT=inv(A*T);
sigma(1:4,1:4)=iAT*diag([epsx,epsx,epsy,epsy])*iAT'; % more stable
if sigp > 0
    D0=periodic_dispersion(Rend);
    sigma=sigma+D0*D0'*sigp^2; % add momentum spread
end
end
```

periodic_dispersion()

The function `periodic_dispersion()` receives the 5x5 transfer matrix R_{end} as input and returns the periodic dispersion D_0 , which obeys $D_0 = R_{end}D_0$.

```
function D0=periodic_dispersion(Rend)
D=(eye(4)-Rend(1:4,1:4))\Rend(1:4,5);
D0=[D;1];
end
```

calculate_dispersion()

The function `calculate_dispersion()` receives the description of the beamline and the initial dispersion D_0 as input and returns the dispersion D and its derivative D_p .

```
function [D,Dp]=calculate_dispersion(beamline,D0)
[Racc,spos]=calcmat(beamline);
D=zeros(length(spos),1); Dp=D;
for k=1:length(spos)
    D(k)=Racc(1,:,k)*D0;
    Dp(k)=Racc(2,:,k)*D0;
end
end
```

tunes()

The function `tunes()` receives the 5x5 transfer matrix R_{end} and returns the horizontal and vertical tunes, Q_x and Q_y , respectively. This routine only works for uncoupled beamlines!

```

function Q=tunes(Rend);
[Qx,alphax,betax,gammax]=R2beta(Rend(1:2,1:2));
[Qy,alphay,betay,gammay]=R2beta(Rend(3:4,3:4));
Q=[Qx,Qy];
end

```

drawmag()

The function `drawmag()` receives the beamline description and the vertical position `vpos` and height of the magnets on the plot as input and produces a graphical rendition of the quadrupoles and dipoles on a plot.

```

% drawmag.m, draw magnet lattice
function drawmag(beamline,vpos,height);
hold on
legend('AutoUpdate','off')
nlines=size(beamline,1);
nmat=sum(beamline(:,2))+1;
spos=zeros(nmat,1);
ic=1;
for line=1:nlines
    for seg=1:beamline(line,2)
        ic=ic+1;
        switch beamline(line,1)
            case 2 % thin quadrupole
                dv=0.15*height*sign(beamline(line,4));
                rectangle('Position',[spos(ic-1),vpos+dv,0.1,height])
            case 4 % sector dipole
                L=beamline(line,3);
                rectangle('Position',...
                    [spos(ic-1),vpos+0.25*height,L,0.5*height])
            case 5 % thick quadrupole
                L=beamline(line,3);
                dv=0.15*height*sign(beamline(line,4));
                rectangle('Position',[spos(ic-1),vpos+dv,L,height])
            case 60 % thin skew quadrupole
                dv=0.3*height; %*sign(beamline(line,4));
                rectangle('Position',[spos(ic-1),vpos+dv,0.1,0.6*height],'EdgeColor',[1,0,0])
        end
        spos(ic)=spos(ic-1)+beamline(line,3);
    end
end
plot([spos(1),spos(end)], [vpos+0.5*height,vpos+0.5*height], 'k:');
end

```

sagrub()

The function `sagrub()` implements the decomposition of a full-turn matrix R according to Sagan and Rubin's paper. It receives a 5×5 full-turn matrix R and a parameter flipped, which, if it is unity, instructs the routine to calculate the flipped solution, as input and returns the 4×4 matrices \mathcal{O} , \mathcal{A} , and T that obey Equation 3.104. The fourth output p , defined in the last line of the function, contains the eigentunes and beta functions as well as other parameters related to coupled beam lines.


```

function [O,A,T,p]=sagrub(R,flipped)
if nargin==1, flipped=0; end
M=R(1:2,1:2); N=R(3:4,3:4); m=R(1:2,3:4); n=R(3:4,1:2);
nplus=[n(2,2),-n(1,2);-n(2,1),n(1,1)];
H=m+nplus;
TRMN=trace(M-N);
if TRMN==0, TRMN=1e-16; end
gamma=sqrt(0.5+0.5*sqrt(TRMN^2/(TRMN^2+4*det(H))));
C=-H*sign(TRMN)/(gamma*sqrt(TRMN^2+4*det(H)));
Cplus=[C(2,2),-C(1,2);-C(2,1),C(1,1)];
AA=gamma^2*M-gamma*(C*n+m*Cplus)+C*N*Cplus;
BB=gamma^2*N+gamma*(n*C+Cplus*m)+Cplus*M*C;
if flipped==1 % calculate the flipped mode
    gammaf=sqrt(1-gamma^2);
    Aaf=C*BB*Cplus/gammaf^2;
    BBf=Cplus*AA*C/gammaf^2;
    C=-gamma*C/gammaf; % update C to flipped version
    Cplus=[C(2,2),-C(1,2);-C(2,1),C(1,1)]; % and Cplus
    gamma=gammaf; AA=Aaf; BB=BBf; % and the others
end
T=[gamma*eye(2),-C;Cplus,gamma*eye(2)];
[Q1,alpha1,beta1,gamma1]=R2beta(AA);
[Q2,alpha2,beta2,gamma2]=R2beta(BB);
if ~isreal(Q1), disp('Mode 1 unstable'); end
if ~isreal(Q2), disp('Mode 2 unstable'); end
A=zeros(4);
A(1,1)=1/sqrt(beta1); A(2,1)=alpha1/sqrt(beta1); A(2,2)=sqrt(beta1);
A(3,3)=1/sqrt(beta2); A(4,3)=alpha2/sqrt(beta2); A(4,4)=sqrt(beta2);
O=zeros(4);
O(1,1)=cos(2*pi*Q1); O(1,2)=sin(2*pi*Q1);
O(2,1)=-O(1,2); O(2,2)=O(1,1);
O(3,3)=cos(2*pi*Q2); O(3,4)=sin(2*pi*Q2);
O(4,3)=-O(3,4); O(4,4)=O(3,3);
p=[Q1,alpha1,beta1,Q2,alpha2,beta2,gamma,C(1,1),C(1,2),C(2,1),C(2,2)];
end

```

dipole_slicer()

The function `dipole_slicer()` receives the description of the beamline and a parameter `nslice` as input. It then replaces each dipole by `nslice` segments, which improves the accuracy for a subsequent calculation of the radiation integrals.

```

function out=dipole_slicer(beamline,nslice)
nlines=size(beamline,1);
out=beamline;
for line=1:nlines
    if beamline(line,1)==4 % dipole found
        out(line,2)=out(line,2)*nslice;
        out(line,3)=out(line,3)/nslice;
        out(line,4)=out(line,4)/nslice;
    end
end
end

```

radiation_integrals()

The function `radiation_integrals()` receives the description of the beamline and the parameter `flipped` as input and returns the radiation integrals for coupled lattices, as discussed in [V. Ziemann, A. Streun, *Equilibrium parameters in coupled storage ring lattices and practical applications*, forthcoming].

```
function [I1,I2,I3,I4,I5]=radiation_integrals(beamline,flipped)
if nargin==1, flipped=0; end
[Racc,spos,nmat,nlines]=calcmat(beamline);
Rturn=Racc(:,:,end);
[O,A,T,p]=sagrub(Rturn,flipped);
D0=periodic_dispersion(Rturn);
rho=zeros(nmat,1);
ic=1; % copied from calcmat
for line=1:nlines % needed because of segmentation
    for seg=1:beamline(line,2)
        ic=ic+1;
        switch beamline(line,1)
            case 4 % sector dipole
                phi=beamline(line,4)*pi/180; % convert to radians
                rho(ic)=beamline(line,3)/phi;
            end
        end
    end
end
I1=0; I2=0; I3=0; I4a=0; I4b=0; I5a=0; I5b=0; % initialize to zero
for k=2:nmat-1 % loop over lattice
    if abs(rho(k)) > 1e-9
        fudge=1;
        if abs(rho(k-1)) < 1e-9, fudge=0.5; end % end segments at start
        if abs(rho(k+1)) < 1e-9, fudge=0.5; end % and exit of element
        ds=spos(k)-spos(k-1); % length of segment
        R=Racc(:,:,k)*Rturn*inv(Racc(:,:,k)); % move FTM to point k
        [O,A,T,p]=sagrub(R,flipped); Sinv=A*T;
        try
            S=inv(Sinv);
        catch
            disp('Warning in radiation_integrals: failed inverse!')
            S=0*Sinv;
        end
        Dxy=Racc(:,:,k)*D0; D=Sinv*Dxy(1:4);
        Ha=D(1)^2+D(2)^2; Hb=D(3)^2+D(4)^2;
        I1=I1+ds*fudge*Dxy(1)/rho(k); % [m]
        I2=I2+ds*fudge/rho(k)^2; % [1/m]
        I3=I3+ds*fudge/abs(rho(k))^3; % [1/m^2]
        I4a=I4a+ds*fudge*(S(1,1)*D(1)+S(1,2)*D(2))/rho(k)^3; % [1/m]
        I4b=I4b+ds*fudge*(S(1,3)*D(3)+S(1,4)*D(4))/rho(k)^3; % [1/m]
        I5a=I5a+ds*fudge*Ha/rho(k)^3; % [1/m]
        I5b=I5b+ds*fudge*Hb/rho(k)^3; % [1/m]
    end
end
I4=[I4a,I4b];
I5=[I5a,I5b];
end
```

Bmags()

The function `Bmags()` receives two 2×2 beam matrices `sig1` and `sig2` as input and returns the mismatch parameter `Bmag`, which is defined in Equation 8.15.

```
function out=Bmags(sig1,sig2)
eps1=sqrt(det(sig1)); beta1=sig1(1,1)/eps1; alpha1=-sig1(1,2)/eps1;
eps2=sqrt(det(sig2)); beta2=sig2(1,1)/eps2; alpha2=-sig2(1,2)/eps2;
out=0.5*(beta1/beta2+beta2/beta1+beta1*beta2*(alpha1/beta1-alpha2/beta2)^2);
end
```