Companion software for "Volker Ziemann, *Hands-on Accelerator physics using MATLAB, CRCPress, 2019*" (https://www.crcpress.com/9781138589940)

Beam optics support functions 4D (Section 3.5)

Volker Ziemann, 211119

In this live script we define the functions for the 4D beam optics calculations, such as `calcmat()` that a frequently used in other calculations. All described functions reside in the subdirectory `4D` that is contained in the archive `BeamOpticsSupportFile.zip`. Any scripts using these function need to include that subirectory with the command "`addpath ./4D`".

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

The following function receives the `beamline` description as input and returns

- Racc(4,4,nmat): transfer matrices from the start to the each 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

```
function [Racc,spos,nmat,nlines]=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
spos=zeros(nmat,1);           % longitudinal position
ic=1;                         % element counter
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(4);            % 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));
       otherwise
         disp('unsupported code')
     end
     Racc(:,:,ic)=Rcurr*Racc(:,:,ic-1);    % concatenate
     spos(ic)=spos(ic-1)+beamline(line,3); % position of element
```

```
    end
  end
end
```

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

The function `DD()` receives the length `L` of a drift space and resturns the 4x4 transfer matrix `out` for a drift space.

```
function out=DD(L)
out=eye(4);
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 4x4 transfer matrix `out` for a thin-lens quadrupole.

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

Transfer matrix for a thick quadrupole `Q(F)`

The function `QQ()` receives the length `L` and `k1` as input and returns the 4x4 transfer matrix `out` for a thick quadrupole.

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

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

The function `SB()` receives the length `L` and bending radius `rho` of a horizontally deflecting sector dipole magnet and returns its 4x4 transfer matrix `out`.

```
function out=SB(L,rho)
phi=L/rho;
out=eye(4);
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)];
end
end
```

Transfer matrix for coordinate rotation `ROLL(phi)`

The function `ROLL()` receives the roll angle `phi` (in degree) around the s-direction as input and returns the corresponding 4x4 transfer matrix `out`.

```
function out=ROLL(phi)  % phi in degree
c=cos(phi*pi/180); s=sin(phi*pi/180);
out=zeros(4);
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 4x4 beam matrix `sigma0` as input an 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,nmat,nlines]=calcmat(beamline);
betax=zeros(1,nmat); betay=betax;
for k=1:nmat
```

```
   sigma=Racc(:,:,k)*sigma0*Racc(:,:,k)';
   betax(k)=sigma(1,1); betay(k)=sigma(3,3);
end
plot(spos,betax,'k',spos,betay,'r-.');
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_sigmas()

The function `plot_sigmas()` receives the `beamline` description and the initial 4x4 beam matrix `sigma0` as input an produces a plot of the horizontal and the vertical beam sizes σ_x and σ_y. It uses Equation 3.43 to propagate σ.

```
function plot_sigmas(beamline,sigma0)
[Racc,spos,nmat,nlines]=calcmat(beamline);
sigmax=zeros(nmat,1); sigmay=sigmax;   % allocate space
for k=1:nmat
   sigma=Racc(:,:,k)*sigma0*Racc(:,:,k)';
   sigmax(k)=sqrt(sigma(1,1)); sigmay(k)=sqrt(sigma(3,3));
end
plot(spos,sigmax,'k',spos,sigmay,'k-.');
xlabel(' s[m]'); ylabel('\sigma_x,\sigma_y [m]')
legend('\sigma_x','\sigma_y')
axis([0, max(spos), 0, 1.05*max([sigmax,sigmay])])
end
```

periodic_beammatrix()

The function `periodic_beammatrix()` receives the 4x4 transfer matrix `Rend` and the emittances `epsx` and `epsy` as input and returns the 4x4 beam matrix σ that obeys $\sigma = R_{end}\sigma R^t_{end}$. In other words, it is periodic.

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

tunes()

The function tunes() receives the 4x4 transfer matrix `Rend` and returns the horizontal and vertical tunes, Q_x and Q_y, respectively.

```
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.

```
function drawmag(beamline,vpos,height)
hold on
% legend('AutoUpdate','off')  % avoids an extra entry for the magnet drawings
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
            dv=0.15*height*sign(beamline(line,4));
            rectangle('Position',[spos(ic-1),vpos+dv,0.1,height])
        case 4
            L=beamline(line,3);
            rectangle('Position', ...
                [spos(ic-1),vpos+0.25*height,L,0.5*height])
        case 5
            L=beamline(line,3);
            dv=0.15*height*sign(beamline(line,4));
            rectangle('Position',[spos(ic-1),vpos+dv,L,height])
    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
```

edteng()

The function edteng() receives a 4x4 full-turn matrix R as input and returns the 4x4 matrices $\mathscr{O}$, $\mathscr{A}$, and T from Equation 3.104. The fourth output `para`, defined in the last line of the function, contains the eigentunes and beta functions as well as other parameters related to coupled beam lines. The algorithm is a straight implementation following reference [13] in the book.

```
function [O,A,T,para]=edteng(R)
TRMN=0.5*(R(1,1)-R(3,3)+R(2,2)-R(4,4));
DETM=R(3,1)*R(4,2)-R(3,2)*R(4,1);
TR=R(1,3)*R(3,1)+R(1,4)*R(4,1)+R(2,3)*R(3,2)+R(2,4)*R(4,2);
CCMU=sqrt(TRMN*TRMN+2*DETM+TR)*sign(TRMN);
if (abs(DETM) < 1E-10 & abs(TR)<1E-10) CCMU=TRMN; end
QQ=TRMN/CCMU;
if (abs(QQ)>1) QQ=0; end
phi=0.5*acos(QQ);
DENOM=CCMU*sin(2D0*phi);
if (abs(DENOM)>1E-10)
  D11=-(R(3,1)+R(2,4))/DENOM; D12=-(R(3,2)-R(1,4))/DENOM;
```

```
  D21=-(R(4,1)-R(2,3))/DENOM; D22=-(R(4,2)+R(1,3))/DENOM;
else
  D11=0; D12=0; D21=0; D22=0;
end
A11=R(1,1)-(D22*R(3,1)-D12*R(4,1))*tan(phi);
A12=R(1,2)-(D22*R(3,2)-D12*R(4,2))*tan(phi);
A21=R(2,1)-(D11*R(4,1)-D21*R(3,1))*tan(phi);
A22=R(2,2)-(D11*R(4,2)-D21*R(3,2))*tan(phi);
B11=R(3,3)+(D11*R(1,3)+D12*R(2,3))*tan(phi);
B12=R(3,4)+(D11*R(1,4)+D12*R(2,4))*tan(phi);
B21=R(4,3)+(D21*R(1,3)+D22*R(2,3))*tan(phi);
B22=R(4,4)+(D21*R(1,4)+D22*R(2,4))*tan(phi);
[Q1,alpha1,beta1,gamma1]=R2beta([A11,A12;A21,A22]);
[Q2,alpha2,beta2,gamma2]=R2beta([B11,B12;B21,B22]);
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=eye(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);
T=eye(4)*cos(phi);
T(3:4,1:2)=[D11,D12;D21,D22]*sin(phi);
T(1:2,3:4)=[-D22,D12;D21,-D11]*sin(phi);
para=[Q1,alpha1,beta1,Q2,alpha2,beta2,phi,D11,D12,D21,D22];
end
```