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

Beam optics support functions 3D (Section 3.4)

Volker Ziemann, 211119

In this live script we define the functions for the 3D beam optics calculations, such as `calcmat()` that a frequently used in other calculations. All described functions reside in the subdirectory `3D` that is contained in the archive `BeamOpticsSupportFile.zip`. Any scripts using these function need to include that subirectory with the command "`addpath ./3D`".

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

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

- Racc(3,3,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(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));
       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 3x3 transfer matrix `out` for a drift space.

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

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

The function `Q()` receives the focal length `F` as input and returns the 3x3 transfer matrix `out` for a thin-lens quadrupole.

```
function out=Q(F)
out=eye(3);
if abs(F)<1e-8 return; end
out(2,1)=-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 3x3 transfer matrix `out` for a thick quadrupole.

```
function out=QQ(L,k1)
ksq=sqrt(abs(k1));
out=eye(3);
if abs(k1) < 1e-6
    out(1,2)=L;
elseif k1>0
    out(1:2,1:2)=[cos(ksq*L),sin(ksq*L)/ksq;-ksq*sin(ksq*L),cos(ksq*L)];
else
    out(1:2,1:2)=[cosh(ksq*L),sinh(ksq*L)/ksq;ksq*sinh(ksq*L),cosh(ksq*L)];
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 3x3 transfer matrix `out`.

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

```
end
end
```

plot_betas()

The function `plot_betas()` receives the `beamline` description and the initial 3x3 beam matrix `sigma0` as input an produces a plot of the beta function. This function assumes that the emittance of sigma0 is 1, or $\det\sigma_0 = 1$, such that $\sigma_{11} = \beta$ is the beta function. 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);
end
plot(spos,betax,'k');
xlabel(' s[m]'); ylabel('\beta_x [m]')
axis([0, max(spos), 0, 1.05*max(betax)])
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
```

periodic_dispersion()

The function `periodic_dispersion()` receives a 3x3 transfer matrix `Rend` as input and returns the 3x1 initial dispersion vector `D0` that repeats after applying `Rend` to it. `D0` is calculated from Equation 3.94.

```
function D0=periodic_dispersion(Rend)
D=(eye(2)-Rend(1:2,1:2))\Rend(1:2,3); % eq. 3.94
D0=[D;1];
end
```

calculate_dispersion()

The function `calculate_dispersion()` receives the `beamline` description and the initial dispersion `D0` as input and returns an array `D` that contains the dispersion along the beam line as well as the positions `spos`.

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