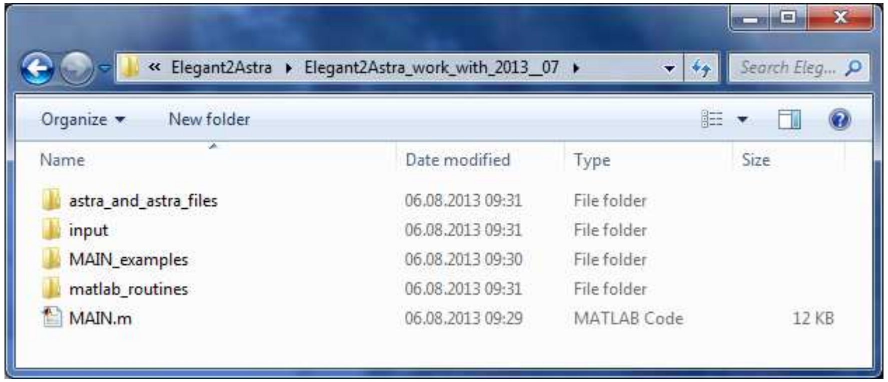
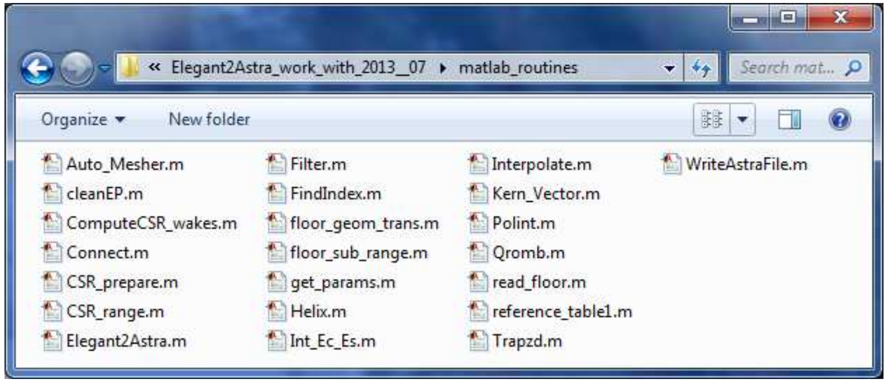
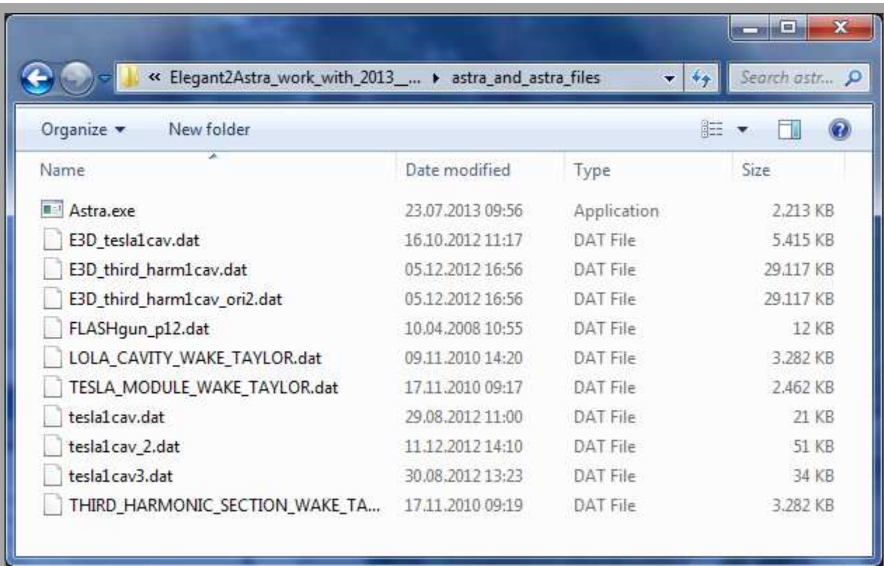
Astra preprocessor

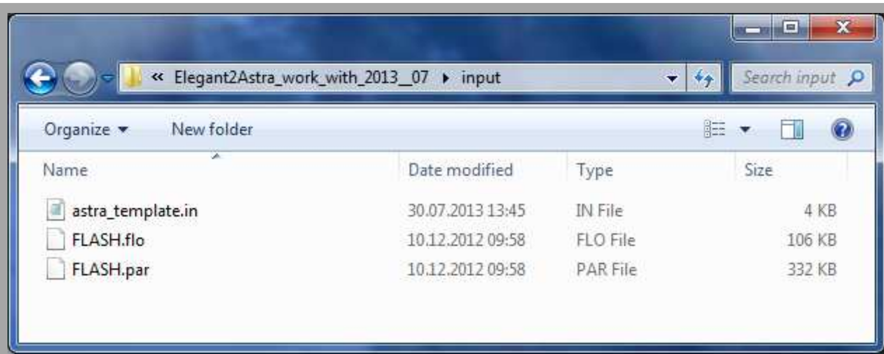
In principle it is possible to simulate a complete bunch compression system with cavities, coupler kicks, bunch compressors, wakes and CSR effects in Astra.

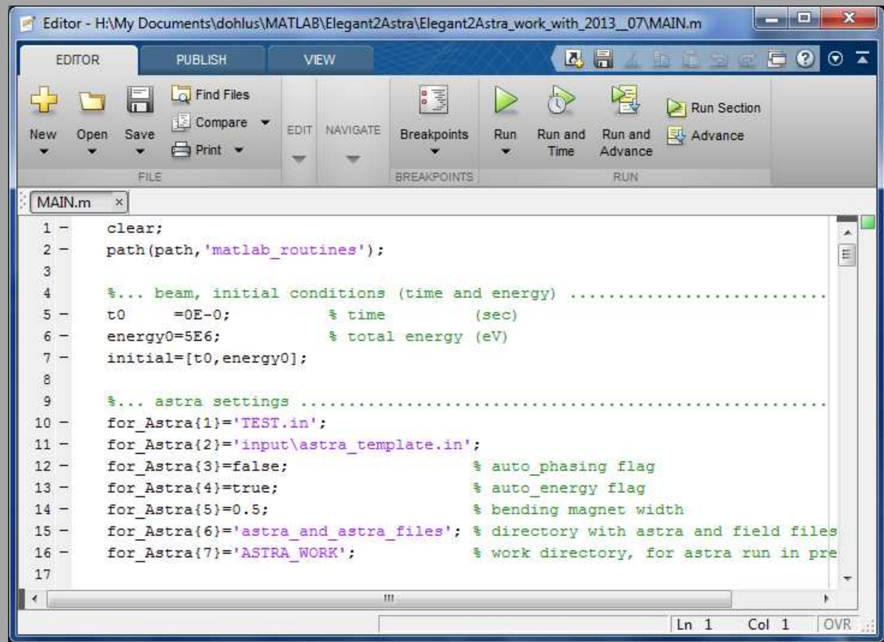
But not in a single run!











Astra executable and input

required: sddsload

SDDS Software Installation Guide for Matlab - Mozilla Firefox

Datei Bearbeiten Ansicht Chronik Lesezeichen Extras Hilfe

SDDS Software Installation Guide for Mat...

www.aps.anl.gov/Accelerator_Systems_Division/Accelerator_Operations_Physics/installationGuide_Matlab.shtml

matlab sddsload

Argonne NATIONAL LABORATORY

Accelerator Operations & Physics

Advance Photon Source
A U.S. Department of Energy, Office of Science,
Office of Basic Energy Sciences national synchrotron x-ray research facility

U.S. DEPARTMENT OF ENERGY | Office of Science

Home | MCR | Operations | Research | Reports | Software | Documentation | Safety & Training | Search APS ...

Argonne Home > Advanced Photon Source > Accelerator Systems Division > Accelerator Operations & Physics

Home
Linux Guide
Solaris Guide
Windows Guide
Pelegant Guide
Matlab Guide
License Agreement
ASD
AES
XSD
APS
Argonne

Guide to Installing SDDS support for Matlab

R. Soliday

Step 1 Linux

Download or build the SDDS Java Binary (JAR) file.

Step 1 Windows

Download the Java SDDS Binaries or build the the SDDS Java Binary (JAR) file.

Step 2 Linux

Edit /usr/local/matlab/toolbox/local/classpath.txt (your location may be different) and add a link to SDDS.jar.

Step 2 Windows

Edit C:\MATLAB6p1\toolbox\local\classpath.txt (your location may be different) and add a link to c:/Program Files/APS/Java SDDS/SDDS.jar

Step 3

If your Matlab version is older than version 13 you will have to install the Java Runtime Environment version 1.3.1. The newer versions will not work with Matlab and the version that comes with Matlab is too old to work with the Java SDDS library. Once installed you will have to set the environment variable MATLAB_JAVA to the directory of the JRE.

Step 4

Download and unpack the SDDS Matlab M-Files. These files can be used by Matlab if they are in the current working directory or if they are added to Matlab's path.

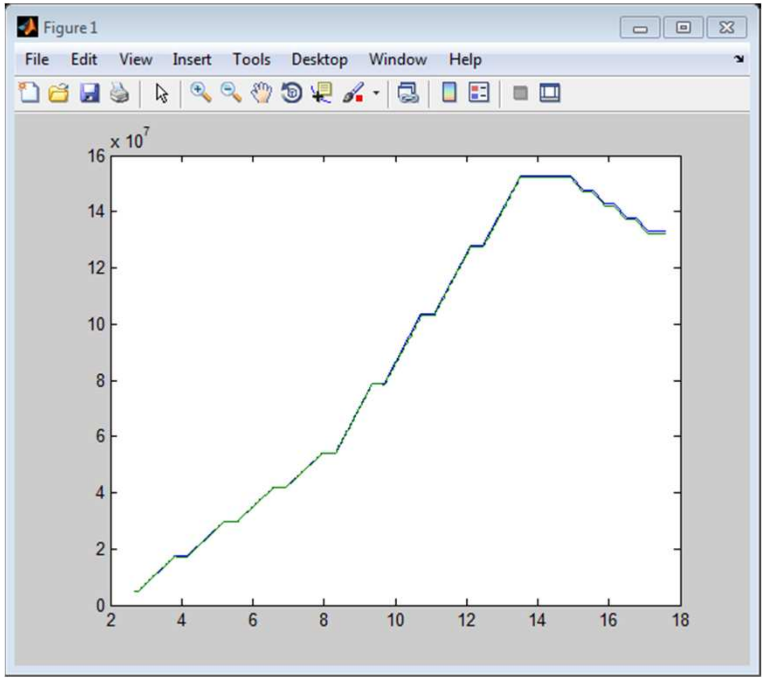
Step 5

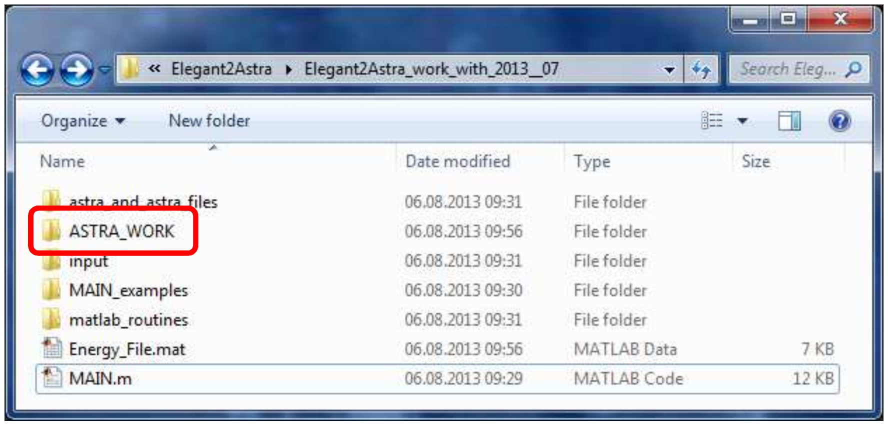
You can now load sdds files into a Matlab structure. The structure looks like:

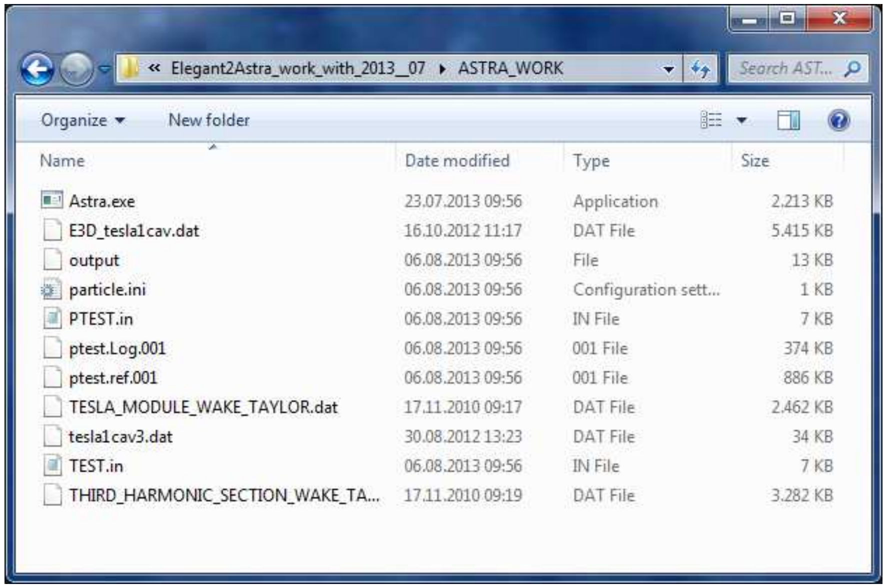
```
sdds.filename
sdds.ascii
sdds.pages
sdds.parameter_names
sdds.array_names
sdds.column_names

sdds.description.contents
                .text

sdds.parameter.[parameter name].type
                               .units
                               .symbol
                               .format_string
                               .description
                               .data

sdds.column.[column name].type
                         .units
                         .symbol
                         .format_string
```

run MAIN.m

```
>> MAIN
C1.ACC1          RFCA          2.9961
C2.ACC1          RFCA          4.3809
C3.ACC1          RFCA          5.7657
C4.ACC1          RFCA          7.1505
C5.ACC1          RFCA          8.5353
C6.ACC1          RFCA          9.9201
C7.ACC1          RFCA          11.3049
C8.ACC1          RFCA          12.6897

...

Q5UND6           QUAD          233.531
Q6UND6           QUAD          233.916
Q9EXP            QUAD          242.917
Q10EXP           QUAD          243.687
Q11EXP           QUAD          244.637
D6DUMP           CSBEND        246.333
Q10DUMP          QUAD          250.1608
Q11DUMP          QUAD          251.4845
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
        1 file(s) copied.
```





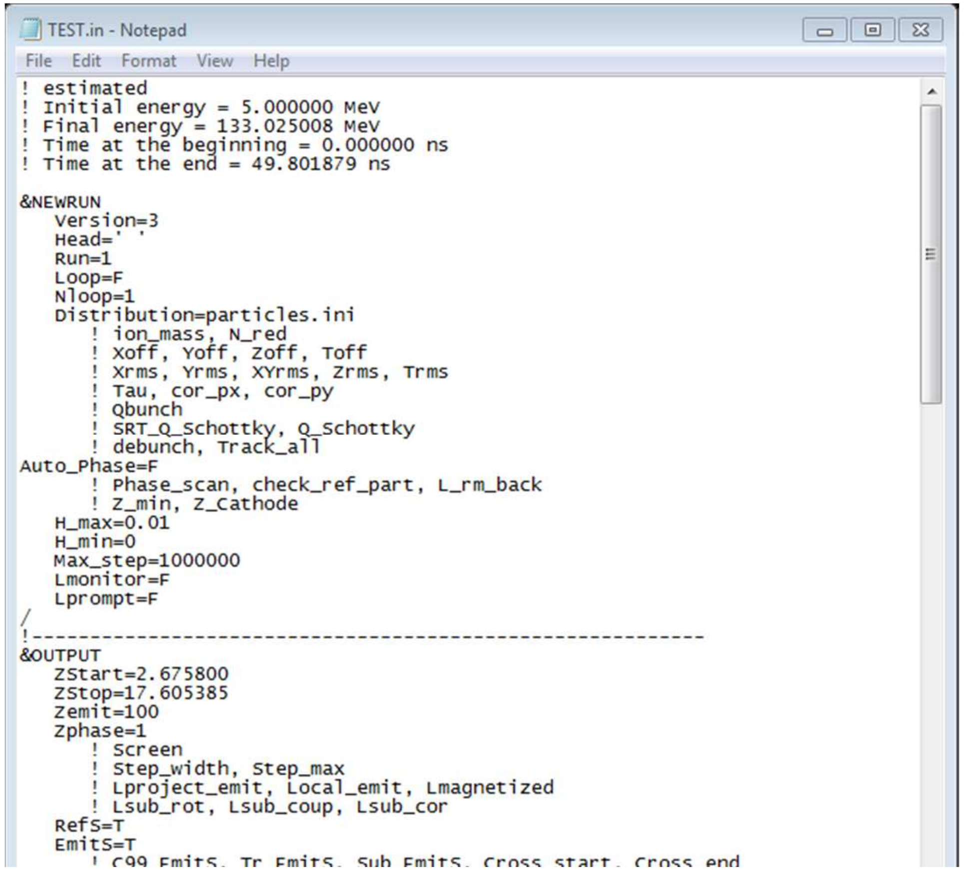


run Astra

copy your particle distribution into ASTRA_WORK
names “ASTRA_WORK“, “particles.ini” are defined in MAIN.m

if required: copy CSR-wakefiles into ASTRA_WORK
see “calculation with CSR wakes”

run Astra with “TEST.in”
name=“TEST.in” is defined in MAIN.m



TEST.in - Notepad

File Edit Format View Help

```
! estimated
! Initial energy = 5.000000 MeV
! Final energy = 133.025008 MeV
! Time at the beginning = 0.000000 ns
! Time at the end = 49.801879 ns

&NEWRUN
   Version=3
   Head=' '
   Run=1
   Loop=F
   Nloop=1
   Distribution=particles.ini
      ! ion_mass, N_red
      ! Xoff, Yoff, Zoff, Toff
      ! Xrms, Yrms, XYrms, Zrms, Trms
      ! Tau, cor_px, cor_py
      ! Qbunch
      ! SRT_Q_Schottky, Q_Schottky
      ! debunch, Track_all
Auto_Phase=F
      ! Phase_scan, check_ref_part, L_rm_back
      ! Z_min, Z_Cathode
   H_max=0.01
   H_min=0
   Max_step=1000000
   Lmonitor=F
   Lprompt=F
/
!-----------------------------------------------------------
&OUTPUT
   ZStart=2.675800
   ZStop=17.605385
   Zemit=100
   Zphase=1
      ! Screen
      ! Step_width, Step_max
      ! Lproject_emit, Local_emit, Lmagnetized
      ! Lsub_rot, Lsub_coup, Lsub_cor
   RefS=T
   EmitS=T
      ! C99_Emits, Tr_Emits, Sub_Emits, Cross_start, Cross_end
```

what is done by the preprocessor?

1) geometry transformation and range selection

2) put elements to position

2) calculate energy profile, based on initial energy, cavity voltages and v=c

3) set magnet strength, either absolute (based on energy profile) or relative

4) Astra run with single (reference) particle and autophasing=true

5) read Astra phases from output file and set absolute phases if required

6) CSR preparation:

    detect trajectory (3d)

    approximate by arcs and lines

    define csr-wake-mesh (automesh)

    define csr-wakes in Astra input file

    calculate csr-wakefiles (MATLAB)

7) prepare work directory with Astra input files, field fieles, (not now: csr-wakefiles)

not necessarily in that order

about input

there are three sources of input (for the pre-processor)

1) MAIN.m	with all pre-processor settings
2) astra_template.in	name defined by for_Astra{2}=… template with many astra settings as H_max, H_min, LSPCH, … some default values can be set by "@" as ! @Q_bore=0.035 it sets the bore-radius of all quadrupole
3) FLASH.flo, FLASH.par	names defined by elegant_floor_name=… and elegant_param_name=… elegant files (in SDDS format) of floor coordinates and parameters

about MAIN.m

pre-processor settings

1) Initial conditions: initial time t0 and energy energy0 of reference particle

2) Astra settings

for_Astra{…}=… file names, directory names, auto_phasing and auto_energy flag

3) Elegant sdds files

elegant_floor_name='input\flash.flo';
elegant_param_name='input\flash.par';

4) Geometry transformation and range

first geometry transformation: geom.trafo_pre
define range to be simulated: geom.start, geom.stop
second geometry transformation: geom.trafo_post

5) Cavity definitions

definition of cavity types: cavity_type_list
list with all cavities in beam line (according to elegant): cavity_list
list with groups of cavities (f.i. modules): cavity_group_list

6) Wakes: to be done

7) CSR wakes

csr_list{…}=… define CSR range and parameters
csr_calc=… control parameters

```
%... astra settings ...........................................................
for_Astra{1}='TEST.in';
for_Astra{2}='input\astra_template.in';
for_Astra{3}=true;                        % auto_phasing flag
for_Astra{4}=true;                        % auto_energy flag
for_Astra{5}=0.5;                         % bending magnet width
for_Astra{6}='astra_and_astra_files';     % directory with astra and field files
for_Astra{7}='ASTRA_WORK';                % work directory, for astra run in preparation
```

for_Astra{1}='TEST.in';	name of Astra input file; will be generated in work directory (from template file)
for_Astra{2}=…;	name of Astra template file
for_Astra{3}=true;	auto energy flag: true --> relative magnet strengths f.i. bending radius; false --> absolute field strength
for_Astra{4}=true;	auto phasing flag: true --> Astra auto-phasing; false --> set absolute phases; these phases are determined by a one-particle test run
for_Astra{5}=0.5;	"horizontal" width (in meter) of bending magnets
for_Astra{6}=…;	directory with astra-executable and field files
for_Astra{7}=…;	name of work directory

```
%... geometry transformation and range ..................................
% 1) first geometry transformation "geom_trans_pre"
% 2) define range to be simulated "start","stop"
% 3) second geometry transformation "geom_trans_post"
Z_SHIFT=0.2978;  g_type=4; g_trafo=[[0;0;Z_SHIFT;0],[0;0;0;0]];
                 s_type=1; s_trafo=[];

geom.trafo_pre ={[g_type,s_type],g_trafo,s_trafo};
geom.start     ={0,'C1.ACC1',0,0.1};
geom.stop      ={0,'C4.ACC39',0,0.5};
geom.trafo_post={[0,1],[],[]};
```

geom.trafo_pre=…;	defines an initial geometry transformation of the setup, defined by the elegant floor description; it is a shift + rotation transformation of the Cartesian coordinates and a shift of the s range (path parameter)
geom.start=…; geom.stop=…;	defines the start-point and end-point of the beam line (after geom.trafo_pre)
geom.trafo_post=…;	defines final geometry transformation of the setup selected by geom.start … geom.stop (after geom.trafo_pre)

definition of geometry transformation (geom.trans_pre, geom.trans_post)

geom.trafo_pre ={[g_type,s_type], g_trafo, s_trafo};

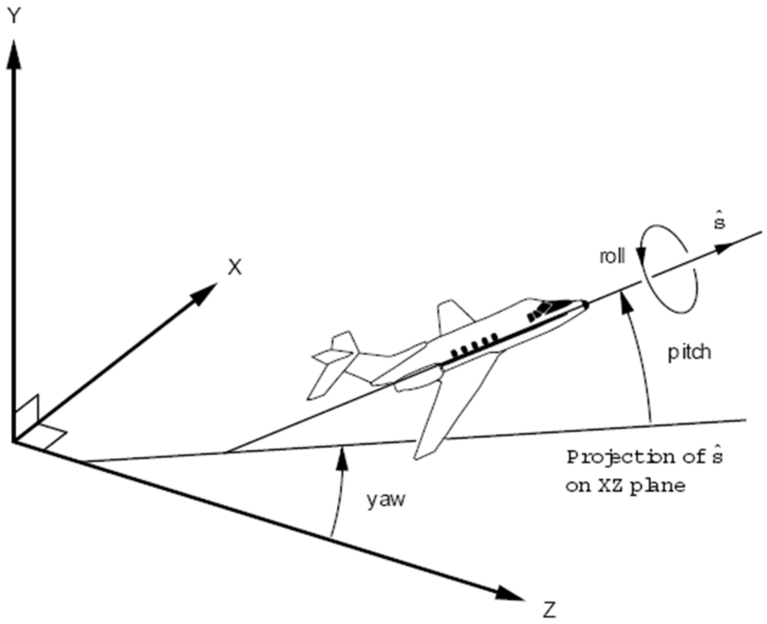
g_type =	0	identity transformation; g_trafo is ignored
	1	"r to 0" shift start point to origin; g_trafo is ignored
	2	"r to 0" + yaw transformation "$\mathbf{e}_x\mathbf{e}_{\|\|}$ to 0"; g_trafo is ignored; (produces warning if $\mathbf{e}_y\mathbf{e}_{\|\|}$~=0)
	3	"r to 0" + pitch transformation "$\mathbf{e}_y\mathbf{e}_{\|\|}$ to 0"; g_trafo is ignored; (produces warning if $\mathbf{e}_x\mathbf{e}_{\|\|}$~=0)
	4	x,y,z-shift + pitch + yaw + roll transformation
	5	roll + yaw + pitch transformation + x,y,z-shift

$$g\_trafo = \begin{bmatrix} x & yaw \\ y & roll \\ z & pitch \\ 0 & L\_radian \end{bmatrix}$$

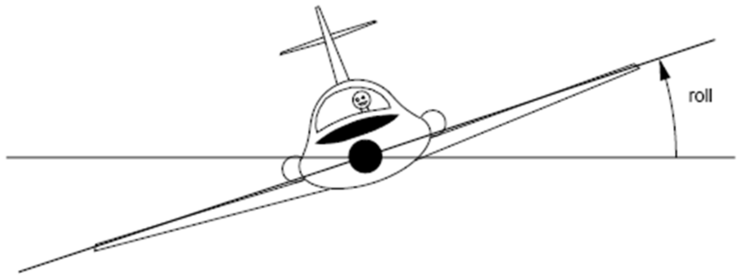
L_radian=true/false -->
yaw, roll, pitch in radians / degrees

s_type =	0	identity transformation; s_trafo is ignored
	1	"s to 0" path parameter starts from zero; s_trafo is ignored
	2	shift s $\longrightarrow$ s+s_trafo
	3	path parameter starts from s_trafo

definition of geometry transformation (geom.trans_pre, geom.trans_post)



The Global Reference System.



The Meaning of the Roll Angle

```
% floor transformation in elegant
% 1) roll  = zrot(psi)
% 2) yaw   = xrot(phi)
% 3) pitch = yrot(theta)
% 4) shift = add [x;y;z]
xrot=@(phi) [1,0,0;0,cos(phi),sin(phi);0,-sin(phi),cos(phi)];
yrot=@(theta) [cos(theta),0,sin(theta); 0,1,0; -sin(theta),0,cos(theta)];
zrot=@(psi) [cos(psi),-sin(psi),0;sin(psi),cos(psi),0;0,0,1];
floor_trafo=@(P,x,y,z,phi,theta,psi) [x;y;z]+yrot(theta)*xrot(phi)*zrot(psi)*P;
floor_trafo_inv=@(P,x,y,z,phi,theta,psi) (yrot(theta)*xrot(phi)*zrot(psi))^-1*(P-[x;y;z]);
```

definition of range (geom.start, geom.stop)

geom.start ={type, parameter1, parameter2, left_shift};
geom.stop ={type, parameter1, parameter2, right_shift};

type =	0	definition by element name (from elegant list)
	1	definition by z coordinate (cartesian coordinate)
	2	definition by s coordinate (path length parameter)

parameter1	element name if type=0 z coordinate if type=1 s coordinate if type=2
parameter2	ignored for type=1 or type=2 number of name appearance (in elegant list) starting from 0; (otherwise some names are not unique!)
left_shift	>0; add drift of this length before start
right_shift	>0; add drift of this length after stop

definition of cavity types: cavity_type_list

```
cavity_type_list(1:N)=struct('type',[],...
                             'frequency',[],...
                             'geom',[],...
                             'param',[],...
                             'wake',[],...
                             'wake_type',[],...
                             'wake_pos',[],...
                             'RZ_field',[],...
                             'RZ_field_param',[],...
                             'E3D_field',[],...
                             'E3D_field_param',[]);
```

list with (at least) all cavities in elegant beam line: cavity_list

```
cavity_list(1:N)=struct('name',[],...
                        'type',[],...
                        'group',[],...
                        'specific',[]);
```

list with groups of cavities (f.i. modules): cavity_group_list

```
cavity_group_list(1:N)=struct(...
                              'group',[],...
                              'specific',[],...
                              'wake',[],...
                              'wake_type',[],...
                              'wake_pos',[]);
```

definition of cavity types: cavity_type_list

```
cavity_type_list(i)=struct('type','TESLA-CAV',...
                           'frequency',1.3E9,...
                           'geom',[xc,yc,zc,zref],...
                           'param',[],...
                           'wake',[],...
                               'wake_type',[],...
                               'wake_pos',[],...
                           'RZ_field','tesla1cav.dat',...
                               'RZ_field_param',[],...
                           'E3D_field','E3D_tesla1cav.dat',...
                               'E3D_field_param',[]);
```

.type cavity-type-identifier;

.frequency resonance frequency in Hz

.geom [xc,yc,zc,zref]; the first three parameters describe the vector from the left on-axis reference point to the center of the cavity; “center of cavity” corresponds to the origin of the cavity field maps; not used: zref is the coordinate of a reference plane (with respect to the cavity origin);

.wake name of wakefield file (if defined);

.wake_type if .wake is defined: type of wakefield (as ‘taylor_method_f’);

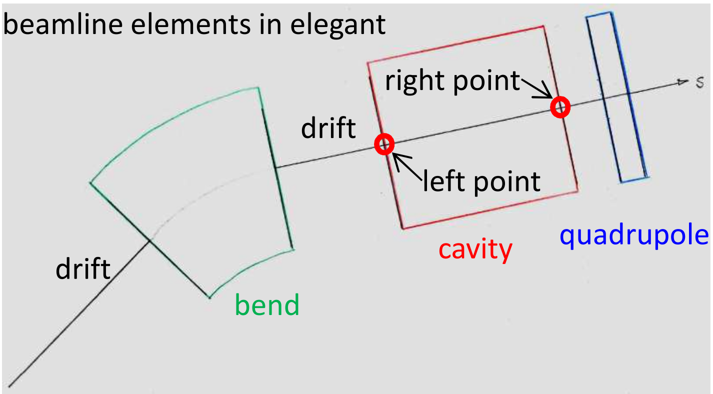
.wake_pos if .wake is defined: [xw,yw,zw,yarot,xarot,zarot]; the first three parameters describe the vector from the left on-axis reference point to the origin of the wake computation; the rotation parameters describe the orientation of the wake-coordinate system (in radian)

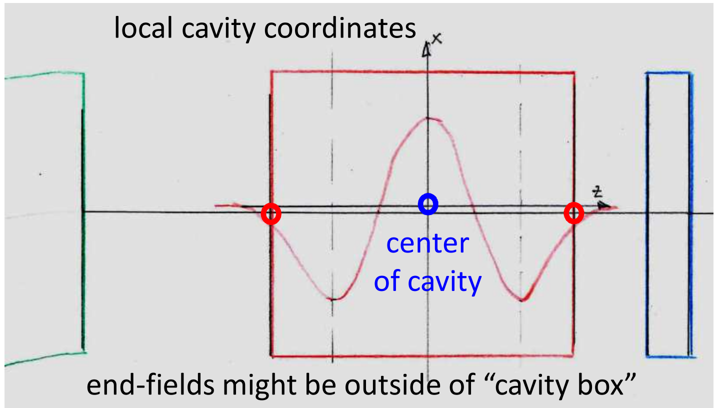
.RZ_field	if defined: name of cavity field file, rz-monopole description;
.RZ_field_param	not used: foreseen for field integral etc.
.E3D_field	if defined: name of cavity field file, E3D description;
.E3D_field_param	not used: foreseen for field integral etc.

the choice to calculate a certain cavity of "cavity_type" with RZ_field or E3D_field is defined in the "cavity" list!

wakes might be defined per cavity (in "cavity_type" list) or per module (in "cavity_group" list)!

files of cavity-fields or wakes have to be present in the directory with the astra-executable and field files (for_Astra{6})





list with (at least) all cavities in elegant beam line: cavity_list

```
cavity_list(i)=struct('name','C1.ACC39',...
                      'type','H3-CAV',...
                      'group','ACC39',...
                      'specific',[am,ph_deg, shift, roto, LE3d]);
```

.name	cavity-identifier, in agreement with unique element-name in elegant list;
.type	cavity-type identifier, in agreement with type-name in cavity_type_list;
.groop	if defined: groop identifier, in agreement with groop-name in cavity_groop_list
.specific	[am,ph_deg,[xs,ys,zs],[yarot,xarot,zarot]],LE3d]; “am” is cavity amplitude in V; “ph_deg” is cavity phase in deg; [xs,ys,zs] is shift of cavity center (by perturbation); [yarot,xarot,zarot] is rotation of cavity (by perturbation); shift after rotation – to my knowledge !!! flag LE3d defines if RZ or E3D format is used;

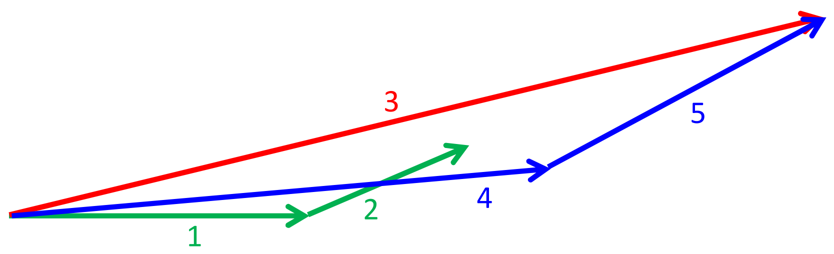
if a cavity is part of a group, the amplitude and phase setting acts relative to the total amplitude and phase, defined for the group list with groups of cavities (f.i. modules): cavity_group_list

```
am=20E6; ph_deg=170.0; shift=[0,0,zw]; roto=[0,0,0];
cavity_group_list(i)=struct('group','ACC39',...
                            'specific',[am,ph_deg, shift, roto],...
                            'wake','THIRD_HARMONIC_SECTION_WAKE_TAYLOR.dat',...
                            'wake_type','taylor_method_f',...
                            'wake_pos',[shift, roto]);
```

.groop	groop identifier, in agreement with groop-names used in cavity_list
.specific	[am,ph_deg,[xs,ys,zs],[yarot,xarot,zarot]]; “am” is groop amplitude in V; “ph_deg” is groop phase in deg; groop amplitude and phase is used if am>=0, otherwise the cavity amplitudes and phases are used (cavity_list); [xs,ys,zs] is shift of cavity center (by perturbation); not used: [yarot,xarot,zarot];
.wake	name of wakefield file (if defined);
.wake_type	if .wake is defined: type of wakefield (as ‘taylor_method_f’);
.wake_pos	if .wake is defined: [xw,yw,zw,yarot,xarot,zarot]; the first three parameters describe the origin of the wake computation with respect to “left of cavity”; the rotation parameters describe the orientation of the wake-coordinate system (in radian)

preprocessor uses no astra-modules:
each cavity is defined individually in the astra-input-file (if it is part of a cavity group or not)

group amplitudes are used if amplitude-values are non-negative



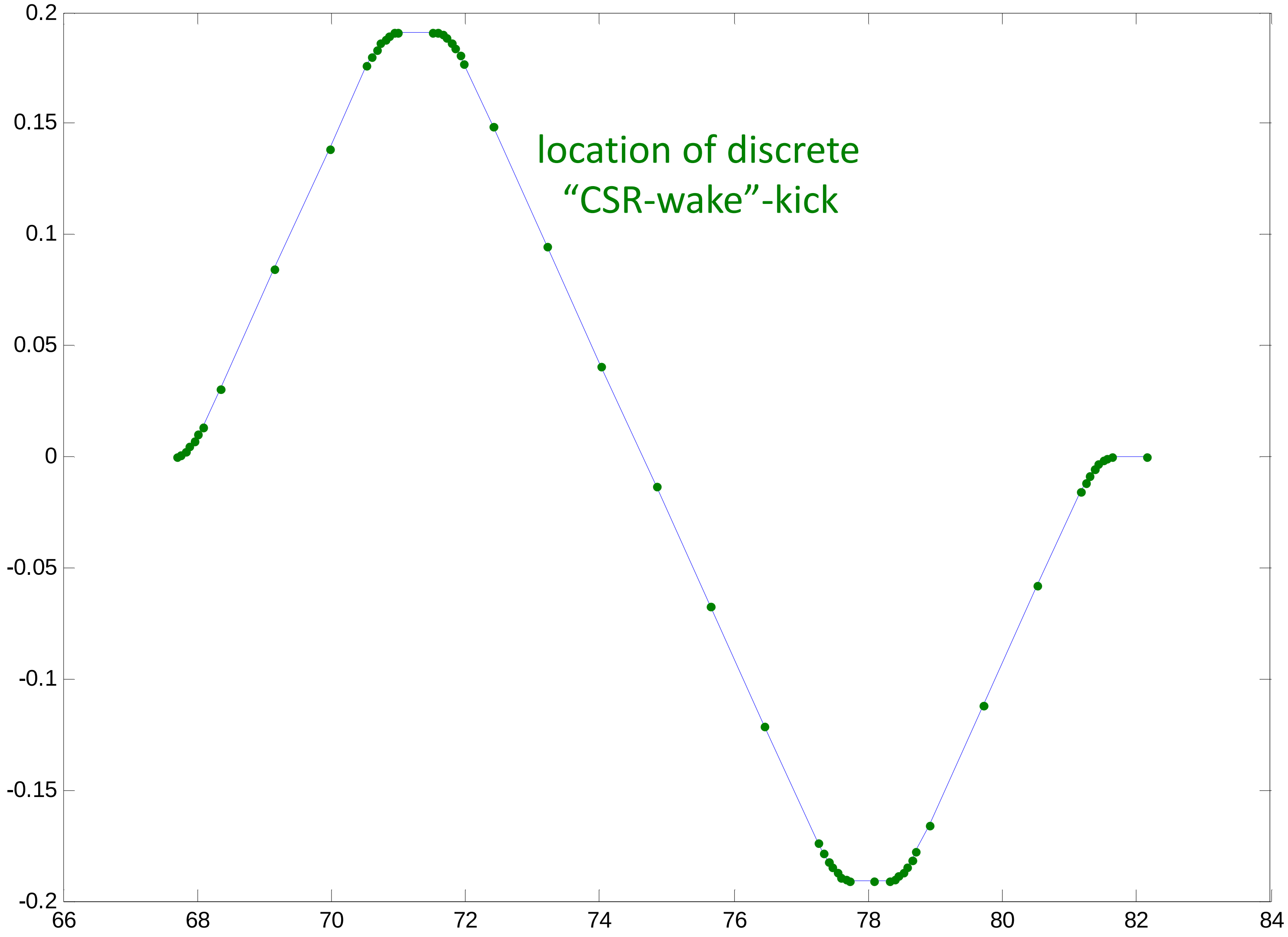
(1) complex amplitude defined for cavity 1
(2) complex amplitude defined for cavity 2
(3) complex amplitude defined for cavity group (cavity 1 + 2)
(4) complex amplitude of cavity 1 in Astra input file
(5) complex amplitude of cavity 1 in Astra input file

interface to impedance data-base

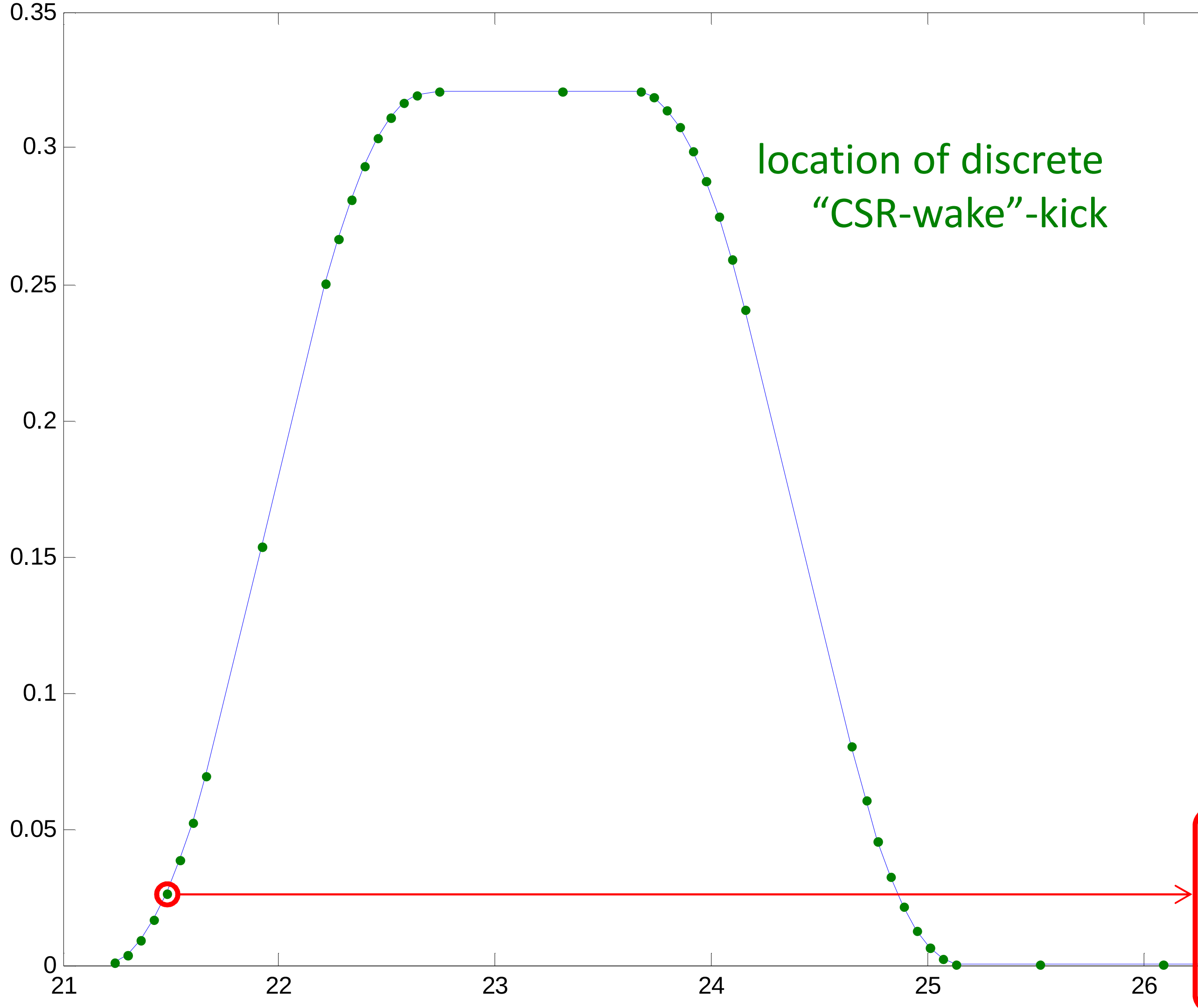
to be defined; wakes might be removed from cavity lists ?!

to be done

trajectory in BC3 (calculated with end fields)



trajectory in BC2 (calculated with end fields)



from Astra input-file:

```
WK_X(1)=0.000843
WK_Y(1)=0.000000
WK_Z(1)=21.238112
WK_ex(1)=0.031020
WK_ey(1)=0.000000
WK_ez(1)=0.999519
WK_hx(1)=0.999519
WK_hy(1)=0.000000
WK_hz(1)=-0.031020
WK_Type(1)='monopole_method_f'
WK_Filename(1)='BC2_wake_1.dat'

WK_X(2)=0.003879
WK_Y(2)=0.000000
WK_Z(2)=21.299322
WK_ex(2)=0.068309
WK_ey(2)=0.000000
WK_ez(2)=0.997664
WK_hx(2)=0.997664
WK_hy(2)=0.000000
WK_hz(2)=-0.068309
WK_Type(2)='monopole_method_f'
WK_Filename(2)='BC2_wake_2.dat'

WK_X(3)=0.009140
WK_Y(3)=0.000000
WK_Z(3)=21.359594
WK_ex(3)=0.105561
WK_ey(3)=0.000000
WK_ez(3)=0.994413
WK_hx(3)=0.994413
WK_hy(3)=0.000000
WK_hz(3)=-0.105561
WK_Type(3)='monopole_method_f'
WK_Filename(3)='BC2_wake_3.dat'

WK_X(4)=0.016651
WK_Y(4)=0.000000
WK_Z(4)=21.419634
WK_ex(4)=0.142665
WK_ey(4)=0.000000
WK_ez(4)=0.989771
WK_hx(4)=0.989771
WK_hy(4)=0.000000
WK_hz(4)=-0.142665
WK_Type(4)='monopole_method_f'
WK_Filename(4)='BC2_wake_4.dat'

WK_X(5)=0.026401
WK_Y(5)=0.000000
WK_Z(5)=21.479349
WK_ex(5)=0.179570
WK_ey(5)=0.000000
WK_ez(5)=0.983745
WK_hx(5)=0.983745
WK_hy(5)=0.000000
WK_hz(5)=-0.179570
WK_Type(5)='monopole_method_f'
WK_Filename(5)='BC2_wake_5.dat'
```

from preprocessor input-file (Main.m):

define CSR ranges, CSR parameters and control parameters:

```
%     BC2
csr_list(1).start={0,'D1BC2',0,0.1};  % start of (1st) CSR range
csr_list(1).stop ={0,'D4BC2',0,2.0};  % end   of (1st) CSR range
csr_list(1).step={0,100};             % parameters for arc-approximation [mode, number of steps or stepwidth]
csr_list(1).file_path='BC2';          % name of directory with csr wakes
csr_list(1).file_name='BC2_wake';     % name of files with csr wake
csr_list(1).path_list=[60 0.7 6];     % parameters of wake-mesh in beam-line coordinate
csr_list(1).bunch_mesh=[1e-6 10000];  % parameters of wake-mesh in bunch coordinate
%     BC3
csr_list(2).start={0,'D1BC3',0,0.1};  % start of (1st) CSR range
csr_list(2).stop ={0,'D14BC3',0,2.0}; % end   of (1st) CSR range
csr_list(2).step={0,100};             % parameters for arc-approximation [mode, number of steps or stepwidth]
csr_list(2).file_path='BC3';          % name of directory with csr wakes
csr_list(2).file_name='BC3_wake';     % name of files with csr wake
csr_list(2).path_list=[60 0.7 6];     % parameters of wake-mesh in beam-line coordinate
csr_list(2).bunch_mesh=[0.25e-6 10000];  % parameters of wake-mesh in bunch coordinate
%     ECOL
csr_list(3).start={0,'D1ECOL',0,0.1}; % start of (1st) CSR range
csr_list(3).stop ={0,'D7ECOL',0,2.0}; % end   of (1st) CSR range
csr_list(3).step={0,100};             % parameters for arc-approximation [mode, number of steps or stepwidth]
csr_list(3).file_path='ECOL';         % name of directory with csr wakes
csr_list(3).file_name='ECOL_wake';    % name of files with csr wake
csr_list(3).path_list=[60 0.7 6];     % parameters of wake-mesh in beam-line coordinate
csr_list(3).bunch_mesh=[0.25e-6 10000];  % parameters of wake-mesh in bunch coordinate

%     control parameters
csr_calc=[0 0 1e-8];  % csr_calc(1) --> prepare astra input file for CSR wakes
                      % csr_calc(2) --> calcualte CSR wakes, if they dont
                      % csr_calc(3) --> precision parameter for arc approximation
```

csr_list defines CSR range and CSR parameters

```
leftshift=0.1; rightshift=2.0; mode=0; Nsteps=100;
NW=60; XW=0.7; nsub=6; ds=0.25E-6; Ns=10000;
%      BC3
csr_list(2).start={0,'D1BC3',0,leftshift};
csr_list(2).stop ={0,'D14BC3',0,rightshift};
csr_list(2).step={mode,Nsteps};
csr_list(2).file_path='BC3';
csr_list(2).file_name='BC3_wake';
csr_list(2).path_list=[NW,XW,nsub];
csr_list(2).bunch_mesh=[ds,Ns];
```

.start	start and stop of CSR range;
.stop	same format as for geom.start and geom.stop; see 4)
.step	{mode,steps}, parameters for arc/line approximation of trajectory; the trajectory is spited into sections with maximal two arcs and one line per section; if mode==1: steps is the number of sections; otherwise: steps is the length of sections
.file_path	name of directory with CSR-wake files;
.file_name	name of CSR-wake files; number and ".dat" will be added;
.path_list	[NW,XW,nsub]; parameters for mesh with wake-positions (beam-line coordinate)
.bunch_mesh	[ds,Ns]; step width and number of steps in wake-file (bunch coordinate)

calculation of CSR wakes:

arc/line approximation of trajectory: $\rightarrow \mathbf{r}_{\text{trajectory}}(S)$

mesh with wake positions: S_k with k = 1 .. NW

auto-mesh parameter: XW
XW=0 $\rightarrow$ equidistant mesh
XW=1 $\rightarrow$ only mesh-points in arcs
(and anything between)

integrated wakes ("kicked" at S_k): $W_k(s) = \int_{S_{k-1}}^{S_k} K(s,S)\,dS$ integrated in nsub steps

CSR-kernel $K(s,S)$ depends on trajectory;
$W_k(s)$ is calculated on the mesh s = [0, ds, … N ds]

control parameters
csr_calc= [LAstra, LCSR, geo_acc];
LAstra=true: prepare Astra input-file with CSR-wakes;
LCSR=true: compute CSR-wakes; (MATLAB $\rightarrow$ wake directory)
geo_acc: accuracy parameter for arc/line approximation

what has to be done?   problems?

1) excessive testing needed

2) interface to impedance data base

2) we need 3d cavity fields (with coupler kicks)

3) Astra: modification of wake-kick algorithm (for wakes in bends)

4) Support of more elements; f.i. sextupoles

known bug (of Astra): wakes of distributions with non-constant charge of macro-particles

known bug of preprocessor: geom.trans_post works only for identity