OPA documentation version 4.1.1

Andreas Streun, June 21, 2025

1 How to use this document

This document is intended to support program developers, who want to maintain the existing OPA code, or to rewrite it in another environment, or to extract useful parts for integration into other codes.

When reading this document, the source code may be examined in parallel, preferably in the Lazarus IDE. In order to see how the code works, OPA may be executed or the tutorial [1] may be consulted. Further information on using OPA is found in the user guide [2] and the underlying physics is outlined in [3].

In this document following styles are used:
Pieces of code are shown in `typewriter style`. `GUI` is a graphical user interface, usually based on class `TForm`. `Frame` if a user-defined GUI-component, usually based on class `TFrame`. `Unit` is a standard Pascal unit.

A GUI or a frame has a list of `actions` corresponding to the event handlers of its components, for example a button to be pressed. The pure Pascal units are passive, they contain a set of public `procedures` which are called from the GUIs. Frames too may contain public `procedures`. Thus in this documentation, action lists are given for the GUIs and frames, and lists of public procedures for the frames and units. All modules may offer public variables, here coloured `x`, `y`, `z` for GUIs, frames, units.

GUI components accept user input and/or launch events. Some have captions, like **Buttons**, others not, like for example a (plot area) responding to mouse clicks. The general scheme is **component** → `event handler`. Event handlers may call private procedures of the GUI or public procedures from units as listed in the `uses` section. For the units `unit`>`procedure` refers to a procedure in a unit and `unit`:`variable` to a public variable..

Red text gives informations about bugs and other problems and/or suggestions for future improvements. Green text mentions particular features which distinguish OPA from other beam dynamics codes and should be preserved.

Questions may be adressed to the author at `mailto:psi@andreas-streun.de`.

2 OPA status

Following semantic versioning [4] version 4.063 of OPA was renamed 4.1.0. However, since up to now only the author worked on the code, the distinctions MAJOR, MINOR, PATCH rather refer to the changes as visible to the users than to the API.

Current version is OPA 4.1.1. It includes 44 Pascal units `.pas`, 28 of them are `GUIs` which are accompanied by a Lazarus form `.lfm` (created by the Lazarus IDE), and 16 of them are Pascal `units`. Five of the GUIs are `frames`, i.e. self-defined components embedded in other GUIs. Four units are located in a parallel folder, here called `../com/` since they are not OPA specific but used with other programs too. Table 1 lists the units. The column "Size" lists the lines of code as a rough estimate for the complexity of the unit. Furthermore, OPA includes a couple of image files `.ico, .bmp` to draw symbols on buttons.

A GUI defines a class, given in table 1 in the column "class". The actual GUI is the one and only instance of this class, i.e. a public variable, usually with the name as the class but without the leading "T". A frame also defines a class but no instance, instead the parent GUI will define one or more instances of this class as variables.

In general GUIs should handle the user interactions only while the Pascal units do all the calculations. This separation was largely realized in OPA, however not strictly. Some minor calculations are done in the GUI in some places.

OPA was developed over more than 30 years along the design of SLS, SLS 2.0 and other machines, and features were implemented as needed for design work. The program code reflects this history and may not present the most logical structure. Also the names of the units (including inconsistent use of capitals) are historical and could be changed to more meaningful descriptors.

In this document section 3 will explain the "backbone" of OPA, which are the global datatypes and variables, the main menu and units for mathematics and graphics which are used by many GUIs. Section 4 contains lattice file handling and the two editors, text-based and "LEGO" block like.

Beam physics starts with section 5 on the interactive linear optics design panel and related GUIs and units and continues with section 6 on off-momentum optics. The non-linear optimization module is explained in section 7, and section 8 covers injection and orbit correction.

Editors for the RF bucket and longitudinal gradient bends are explained in section 9, the three tracking modules with their library in section 10, and everything else like machine layout and magnet currents in section 11.

Table 1: OPA units overview

Name	class	Purpose	Size
opamenu	TMenuForm	the main menu	901
OPAglobal		global variables and procedures	2986
mathlib		mathematical library	2075
../com/Vgraph	Vplot	real number graphics library	1276
../com/asfigure	TFigure	general plot procedure	314
../com/conrect		contour plot calculation	332
../com/asaux		little helpers	349
opalatticefiles		file reading and writing	1839
texteditor	TFormTxtEdt	text editor for lattice file	190
OPAEditor	TFormEdit	interactive lattice editor	346
EdElCreate	TEditElemCreate	creation of an element	144
EdElSet	TEditElemSet	edit element parameters	861
EdSgSet	TEditSegSet	edit segment parameters	444
opticview	Toptic	linear optics design	1143
knobframe	TKnob	slider to set element property	421
Opticstart	Tstartsel	setting for start parameters	615
OpticEnvel	TsetEnvel	setting for optical functions plot	372
OpticTune	TtuneMatrix	adjustment of lattice tune	251
OpticWOMK	TWOMK	write optics markers	125
opticplot		linear beam dynamics calculations	2840
OPAElements		element propagation calculations	2387
OpticMatch	TMatch	linear optics matching	1457
OpticMatchScan	TsetMatchScan	parameter scan	119
OPAtune	TTunePlot	show the tune diagram	634
OPAmomentum		momentum dependence and optimization	949
MomentumLib		momentum dependent calculations	480
OPAChroma		nonlinear optimization	1413
CSexLine		controller for nonlinear element	454
CHamLine		bar indicator for Hamiltonian mode	416
ChromGUILib1		little helper to pass results handles	79
ChromGUILib2		little helper to pass element handles	93
OPAChromaSVector		show Hamiltonian as complex vector	145
chromlib		nonlinear dynamics calculations	1813
OPAorbit		orbit correction and injection	2353
Bucket		plot RF bucket	640
LGBeditor		longitudinal gradient bend editor	702
LGBeditorLib		longitudinal gradient bend calculations	471
OPAtrackP		phase space tracking	1422
OPAtrackDA		dynamic aperture tracking	1666
OPAtrackT		Touschek tracking	1726
tracklib		particle tracking calculations	1502
OPAGeometry		geometric machine layout and matching	2010
OPACurrents		export magnet currents to machine control	303
opatest		test of new features	472

3 Main programs and general use

3.1 Main program

opa.lpr

The main program file which allocates the units as described below and creates some of the forms. It is created more or less automatically by the Lazarus IDE.

3.2 Global data

OPAglobal

This unit defines global constants, types and variables and thus is OPA's "database". It is also initialized first when OPA starts and sets environment variables and default values. Public variables defined here are used by the other modules, since almost all of them use `OPAglobal`. These "global" variables contain the basic lattice data, status flags, handles to access GUI components and temporary data to be exchanged between modules. This unit also provides many public procedures for various non-physics tasks. Only an overview can be given here, and the most important or complex things will be explained. Everything else should become clear from the code, hopefully.

The `OPAglobal` unit grew large large and heterogenous over the years and should be split into several units for the different functions to be performed, or procedures should be moved to other units if applicable.

Like other beam dynamics programs OPA works on a lattice file. It contains *elements*, mainly magnets of different type, and *segments*, which are line-ups of elements and segments. One of the segments is selected and expanded to become *the lattice* to work with. Element parameters, e.g. quadrupole strength in the lattice file can be numbers or arithmetic expressions using *variables*, which are part of the lattice file too.

Uses: `../com/mathlib`, `../com/asaux`

Public constants, types and variables

Constants include general settings, array sizes, color definitions, name strings and values of flags later to be referenced by name.

Following type definitions may be considered as non-trivial:

`ElementType` is a case-record for defining element parameters such as length, quad focusing strength, dipole bending angle etc. Several (but not all) parameters may be arithmetic expressions instead of numbers. For more details see the user guide [2] or the source code.

variable_type is for variables defined by name and value or arithmetic expressions. Variables are referenced in arithmetic expressions of element parameters or in other variables.

AbstractEleType represents either an element or a segment and contains pointers to previous and next abstract elements. The abstract element may be inverted or repeated.

SegmentType is for segments, which are series of abstract elements, defined by pointers to its initial and final abstract element.

LatticeType is a lattice entry. The lattice is built by expanding a segment, so it contains only elements, and additional information on direction (if the element is inverted), and data relevant for orbit correction (misaligmnets and if the element sits on a girder).
Unlike several other codes OPA handles correctly nested inversions of segments to obtain the correct orientation of the element in the lattice. This is relevant for bending magnets with unequal edge properties.

OmarkType is a set of optics data as property of the element type "optics marker".

OpValType is an extended set of optics data used for propagation in OPAElements.

GirderType describes a girder by first and last lattice entry of elements sitting on the girder, and by the type of connection to adjacent girders.

CurveType is a point in an optical functions curve including a pointer to the next point.

CurvePlotType defines a curve by a pointer to its initial point.

DefaultType defines a user setting by name and value. Default values are defined by OPAglobal>Initialization but can be modified by the user and saved.

Other types are mainly shortcut definitions to gather interrelated data.

Here are the most important global variables. If not mentioned otherwise, the type name is the variable name with suffix type. Usually only one instance exists:

Elem and Ella are arrays of Elementtype, where the first one is read from file and manipulated in the editors, while the second one are a subset of elements used in the lattice and to be modified in the various calculations. After terminating a calculation the user is asked to save or cancel the changes, which causes the Ella-data to be copied back to the Elem array or not.

Segm, Lattice and Girder are arrays of corresponding types. Maximum array length is defined in the const section of the unit.

Glob saves some global parameters like beam energy, default aperture size etc.

Status is a group of status flags to be set by the outcome of calculations in order to enable or disable buttons and to re-use data in other calculations.

MainButtonHandles provides global control of the various options in opamenu to enable or disable them based on the status flags.

GlobDef and Def of DefaultType are arrays of user-defined settings, which are read at start and when switching the working folder.

Beam saves beam parameters like tune, emittance etc.

SnapSave stores intermediate results in order to re-use them in other modules, for example

analytical results from the nonlinear optimization to compare them later with tracking. (Furthermore, some data are saved which are relevant for machine control and to be exported by `OPACurrents` to an EPICS `.snap` file.) Data are not invalidated yet, if something was modified in the lattice.

There are more variables of basic types, which better should not be public to avoid errors and confusion. Wherever possible they may become local, or they should be hidden behind set- and get-procedures.

Public procedures

This listing of procedures is more or less inverse to the (historical and illogical) arrangement in the source file:

`Initialization` sets some global variables to reasonable initial values, sets all status flags to false, and sets default values for ∼250 user settings. Then it sets the OPA directory and tries to read the files `opa4_path.ini` which contains a list of last used files. Then it reads `opa4_glob_ini` containing global user settings (at the moment this is only the amount of output). Finally, it takes the folder from the first entry in the list of last used files to set the working folder and reads the file `opa4_set.ini` in this folder to restore the ∼250 user settings. If these files are missing, the default values are used.

`GlobDefWriteFile` and `DefWriteFile` write the user settings back to these files if the session is regularly terminated by **opamenu** → `ExitOPA`.

`MakeLattice` builds the lattice from elements and segments. Since a segment istelf contains elements and segments, the internal procedure `SegLat` is called recursively to unpack the data. The elements, which are used in the lattice are copied from the `Elem` to the `Ella` array, and correctors and monitors with generic names `CH,CV,MON` are expanded into separate instances named `CH001,CH002...` to adress them individually in orbit correction in unit `OPAorbit`. Finally status flags are set.

`GirderSetup` evaluates the girder structure if contained in the lattice and allocates the lattice elements to the corresponding girders.

`IniElem` sets default values for new elements.

`AppendAE, ClearSeg, NAESeg` are procedures to handle segments, which are series of pointers to elements or other segments.

`AppendCurve, ClearCurve` prepare data sets for plotting curves of optical functions.

`AppendChar` builds a linked list of characters for `texteditor` (wrong place).

`putkval, getkval, putSexkval, getSexkval` functions are shortcuts to set or get the parameter which is considered the strength of a magnet.

`FillComboSeg` fills the list of last used files in `opamenu`.

`OPALog` writes a message to the log window in `opamenu`.

`MainButtonEnable` enables or disables the options in `opamenu` depending on the status flags. For example, tracking is only enabled if a periodic solution exists.

PassMainButtonHandles and passErrLogHandle take handles to the GUI components of opamenu to enable other units using OPAglobal to enable/disable them.

EllaSave and Elcompare check if elements in lattice (Ella) have been changed with regard to the original elements (Elem) and ask the user if the changes should be saved.

The procedures and functions listed under "Calculator" are an arithmetic evaluator adapted from [15]. Other procedures not listed here include variable, element and segment to string conversion and other utilities which may be self-explanbatory.

3.3 Menu

opamenu

The main GUI for reading and writing files and to launch the other GUIs. It includes message (log) and status windows. It is always open in the background. Closing it exits OPA.

Uses: OPAglobal, OPAEditor, texteditor, opalatticefiles, opticview, OPAtune, OPAmomentum, OPAChroma, OPAtrackP, OPAtrackDA, OPAtrackT, OPAorbit, OPAGeometry, LGBeditor, Bucket, OPACurrents, opatest

Actions

FormCreate called at start creates the GUI and fills the menu items with data: the list of last used files is established and the status labels are set. Then handles to most components are passed to OPAglobal to make them available to all GUIs, in order to send messages to the log window, update the status in the status window and enable or disable options depending on the result of calculations. Note, that the initialization of OPAglobal is executed first to provide the required information.

File Menu

New → fi_new delete all data to start new lattice from scratch.

Open... → fi_open read OPA lattice file or try to read a lattice file from Tracy2, Tracy3, MAD-X, elegant or BMAD. The file name is displayed as "active file".

Last Used ▶ → fi_last select file from list of last used files

Save, Save as... → fi_[save,svas] save OPA lattice file with old or new name.

Export to ▶ → ex_[tracy2,tracy,madx,elegant,bmad,opanovar] save as file for other programs. Last option expands all arithmetic expressions in lattice and saves as OPA file.

Exit → fi_exit save all settings from session and exit OPA (just closing the GUI will exit OPA without saving settings).

Edit Menu

Text Editor → ed_text launch lattice file text editor texteditor.

OPA Editor → ed_oped launch interactive "LEGO block" editor OPAEditor.

Design Menu

Linear Optics → ds_opti launch interactive linear design opticview.
Off-momentum Optics → ds_dppo launch momentum dependent optics OPAmomentum
Non-linear Dynamics → ds_sext launch non-linear optimizer OPAChroma
Orbit Correction → ds_orbc launch orbit & injection panel OPAorbit
Injection Bumps → ds_injc launch orbit & injection panel OPAorbit
RF Bucket Viewer → ds_rfbu launch RF bucket visualization Bucket
Geometry Layout → ds_geo launch geometry layout and matching OPAGeometry
LGB Optimizer → ds_lgbo launch editor for longitudinal gradient bends LGBeditor

Tracking Menu

Phase Sapce → tr_phsp launch phase space tracking OPAtrackP
Dynamic Aperture → tr_dyna launch dynamic aperture tracking OPAtrackDA
Touschek Lifetime → tr_ttau launch Touschek lifetime tracking OPAtrackT

Extra Menu

Output ▶ → tm_di select the amount of output provided in the log window.
The implementation is incomplete and inconsistent: some output goes to the log window, some to a terminal console (only visible if it is open) and some to a file diagopa.txt, which is created (but never closed...).
Magnet Currents → tm_cur launch panel to calculate magnet currents OPACurrents
The other menu items in this group are temporary tests calling procedures from unit opatest and not relevant to be documented here.

(Segment name) → ComboSeg is a drop-down list of the available segments. The one selected is expanded to become the lattice, calling OPAglobal>MakeLattice.
Show lattice expansion → butlatsh displays the expanded lattice in the log window.
Print → ButLogPrt prints the content of the log file to LogPrint.txt in the working directory.
Clear → ButLogClr clears the log window.

3.4 Mathematics library

mathlib

A library of mathemical functions: definition of types and operations with vectors, matrices

and complex numbers. It further contains some special functions, among them the Touschek integral function, and implementations of Powell's minimizer, LU decomposition and Singular Value Decomposition taken from [13], but extended to dynamic array size. OPA does not link external libraries but all algorithms needed are included in the code.

Uses: none

Public constants, types and variables

Types define vectors and matrices for beam dynamics, geometry and for the Powell minimization procedure. The only public variables are the parameters for Powell.

Public procedures

Most procedures are elementary operations on vectors, matrices and complex numbers, or for special functions, and don't need to be explained.

`CTouschek` and `CTouschek_pol` solve the Touschek lifetime integral, see appendix C.1 in [3] for details. The first procedure solves the integral based on Simpson's rule, the second procedure uses a polynomial approximation for speed-up.

`EulerAng` calculates rotation angles from a rotation matrix as explained in [5].

`LUDCMP` and `LUBKSB` peform LU-decomposition and backsubstitution to solve linear systems of equations as described in [13]. This is used in particuar for matrix inversion, `MatInv` and `MatDet`. The procedures are set fix to 5 dimensions.

`svdcmp` and `svbksb` perform singular value decomposition and backsubstitution to solver lineaer systems of equations as described in [13]. SVD is superiour to LUD for non-square and degenerate systems. An packing/unpacking algorithms was added to use the procedures with arbitrary dimension. The procedure is used in orbit correction in `OPAorbit` and for setting octupole families in `OPAChroma`.

`Powell` is an implementation of Powell's minimizer for steepest gradient search in N dimensions taken from [13]. It is used for sextupole optimization in `OPAChroma`, for longitudinal gradient optimization in `LGBeditor`, and (test mode only) for non-linear optimization in `OPAmomentum`.

3.5 Graphics library

../com/Vgraph

Class `Vplot` based on `TObject` contains a Lazarus `TCanvas` object. Plot commands for `TCanvas` use integer screen pixel coordinates. `../com/Vgraph` wraps these standard plotting commands with procedures of same name but accepting physics coordinates as real numbers. Data are either scaled and forwarded to `TCanvas` or written to an Ecapsulated postscript file `*.eps`.

Furthermore procedures for crating nice axes, for drawing circles and ellipses and for grabbing the screen image have been added.

Uses: none

Public constants, types and variables

There are no public variables. A GUI will define one or more variables of class `Vplot`

Public procedures

`Create` accepts a handle to a `TCanvas` object, which is needed to construct a `Vplot` object. Further some initialization is done.

`PS_start` opens an `*.eps` file of name `psfile` supplied by the calling procedure for output and sets the private flag `PS` to `true` to direct the output to the file and not to the screen. A file header is written, defining the BoundingBox as the canvas screen size. Then macros for text alignment are defined in postscript language. If opening the file fails, an error message is returned.

`PS_stop` closes the `*.eps` file and sets the flag `PS` to `false`.

`SetRange*`, `SetMargin*` etc set plot ranges and scaling as displayed in Fig.1. The vertical screen coordinate counts downwards whereas the physical coordinates as well as the Postscript coordinates count upwards.

`getpx,getpy` and `PS_getpxr,PS_getpyr` translate physical coordinates x, y into screen, resp. Postscript coordinates (upper/lower line):

$$\texttt{px} = \texttt{px0} + \frac{\texttt{pxrange}}{x_{\text{range}}}(x - x_{\min}) \qquad \texttt{py} = \left\{ \begin{array}{r} \texttt{py0+} \\ \texttt{pytotal} - \texttt{py0-} \end{array} \right\} \frac{(-\texttt{pyrange})}{y_{\text{range}}}(y - y_{\min})$$

`AdjustAspectRatio` adjusts the plot ranges such, that the unit is the same horizontal and vertical, i.e. a circle appears as circle not as an ellipse.

`Axis` plots an axis with even numbers as annotations and ranges, also extracting an exponent if needed.

`GetAxisSpace` returns the space required for the axis annotations without drawing the axis in order to adjust the plot range accordingly. It is called first by the calling GUI, in particular by `../com/asfigure`.

`Circle` and `Ellipse` plot a circle or a (sheared) ellipse in beam dynamics notation.

`GrabImage` copies the screen canvas to the system clipboard, useful to catch plots for draft notes.

All the other procedures are mainly wrappers for the standard plot commands or shortcuts to draw arrows, symbols etc.

`Stroke` does nothing on the screen but terminates a series of plot commands in Postscript. Curve plotting procedures need to terminate a `LineTo...` loop by calling `stroke`.

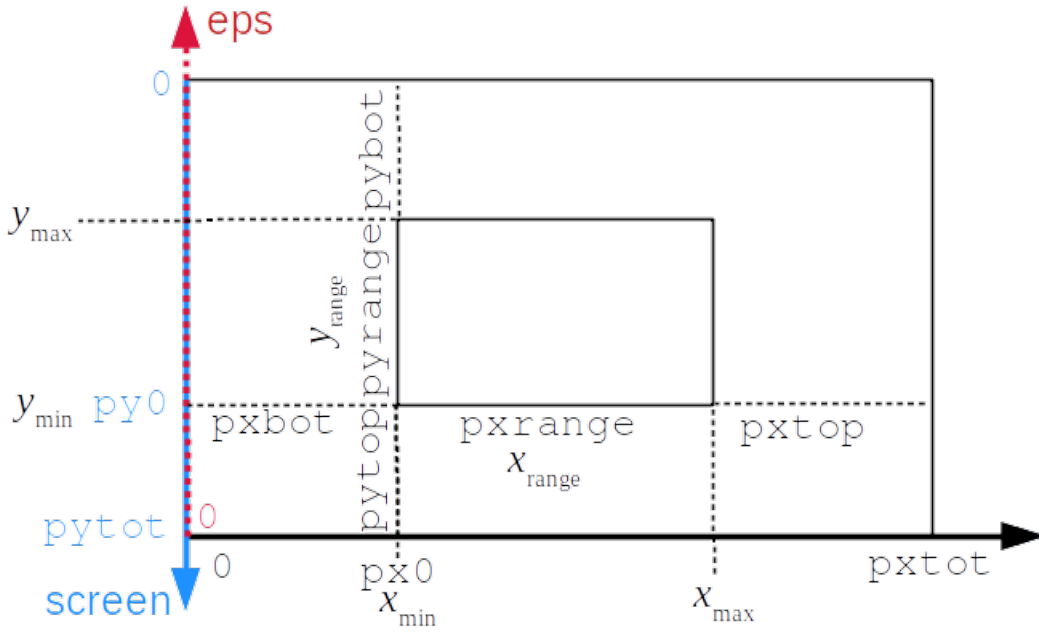

Figure 1: `../com/Vgraph` coordinates

Direct EPS export is a convenient function to immediately create high resolution graphics ready for publication in a LaTeXdocument.

3.6 Plot area

`../com/asfigure`

This frame contains a paintbox (i.e. a plot area) to be embedded in a GUI and a `Vplot` object using the paintbox canvas. Added functionality includes handles to edit fields in order to translate mouse actions to numbers.

Uses: `../com/Vgraph`, `../com/asaux`

Public constants, types and variables

`plot` is a variable of class `Vplot` in `../com/Vgraph`. It is accessible to the parent GUI for direct plotting.

Public procedures

`assignScreen` creates `Vplot` and passes the paintbox canvas in order to plot on the screen.

`forceMargin*` set fixed margins for the plot and supresses auto-scaling by subsequent calls to `Init` or `Vgraph`>`Axis`.

`Init` starts the plot. Inputs are the horizontal and vertical ranges, flags where to put axes and the descriptions of the axes, a flag to mark the origin and another flag to adjust the aspect ratio (i.e. increasing the smaller of the two ranges until the units are of same size, see `Vgraph`>`AdjustAspectRatio`).

`SetSize` to be called from the parent GUI on resizing is used to change the size of this frame.

`PassEditHandle*` accepts the handle to a `TEdit` field in order to link it to the mouse position inside this frame.

`PassFormHandle` accepts a handle to the parent GUI. (?)

`GetRange` returns the range of a rectangular region in physical coordinates, which has been marked by dragging the mouse inside the plot.

`UnfreezeEdit` frees the edit fields again, when it had been frozen by an Mouse/up event (see below).

Actions

(plot area) → `pMouseDown,pMouseMove,pMouseUp` handles mouse events inside the plot area. When edit fields are linked, the physical coordinates of the plot are displayed in the fields while moving the mouse. On releasing the mouse, the edit field freezes, i.e. saves the last data and changes its color. This is used in `OPAtrackP` to see the particle coordinates corresponding to the cursor position.
When the mouse is dragged, the region between mouse down and up is used to rescale the plot area. This is used to zoom in in `OPAGeometry` – This works in Windows but not in Linux, probably due to different event handling – not yet understood.

(plot area) → `pDblClick` grabs the plot image to the Clipboard on double-clicking the mouse.

3.7 Contour plot calculation

../com/conrect

This contour plot calculation is taken from [14] with minor adaptations. Given a 2D array of rectangular gridded data it returns an array of contour lines.

Uses: none

Public constants, types and variables

`Con...` are a set of types to store the data. `ConLinesType` contains points of a straight line belonging to an contour of height index `ih`, and `ConLinesArray` is an array of these.

Public procedures

Conrec is the only procedure, calculating the contours.

3.8 Utilities

../com/asaux

This unit is a collection of "little helpers" for formatting numbers and strings, for defining colors etc. Some of this functions may be obsolete meanwhile, since newer versions of Lazarus and Pascal may include corresponding functions. Procedures are short and probably self-explaining. Not all of them are used in OPA (the ../com folder units are used by several programs).

Uses: none

4 Lattice I/O and Edit

4.1 Lattice files

opalatticefiles

Lattice data are read from and saved to files called *.opa. Reading a file copies its content into a linked list of characters (basically a long string) in opamenu>ReadFile.

There is an option to read additional files of magnet calibrations and allocations (i.e. real magnet types and names associated with the element), in order to prepare data for the machine control system.

Files from other beam dynamics programs can be read and written, however, this includes only the basic parameters and may require some manual editing to make these files work. This includes Tracy-2 or -3 files *.lat, Elegant files *.ele,*.lte, MAD files *.mad and MAD sequence files *.seq.

Uses: OPAglobal, mathlib, ../com/asaux

Public procedures

Latread evaluates the character list of type TextBuffer: comments, enclosed by { } brackets, are skipped. An input line is terminated by a semicolon (;), its length is unlimited and it may break over several lines in the lattice file. Input lines are evaluated by searching first for a colon (:) which identfies an element or segment, or, if not found, a variable.

A variable input line contains a name, an equal sign (=) and a value or an expression. Reserved variables are found searching for the names TITLE, ALLOCATION, CALIBRATION (containing a title of the lattice and the names of optional allocation and calibration files), and by searching in the list of keywords globkeyw containing ENERGY and other global parameters. Other names not matching these keywords are considered as user-defined variables.

An element or segment input line contains a name, the colon (:) and a comma-separated list of tokens.

An element is recognized if the colon is followed by a valid element type name in list elemkeyw (which is based on ElemName in OPAglobal). Then the tokens are expected to contain each a parameter identifier, an equal sign (=) and a value or expression. There are many parameters for the different element types. The element is created and the values are assigned.

For a segment the tokens are names of elements or segments, both are assigned to a linked list of type AbstractEleType as defined in OPAglobal.

If names of allocation or calibration files were given, these files are read now.

Finally, the lattice is built from the last segment calling OPAglobal>MakeLattice.

LatReadCom is called after LatRead and searches the lattice string for a pair of tokens {com and com} to save the text between. All other comments contained in the lattice and enclosed in { } brackets only are not saved.

ReadAllocation and ReadCalibration read additional optional files containing lists of magnet types and real names and the magnet calibrations in order to calculate the magnet currents in OPACurrents. An example for SLS is found at https://ados.web.psi.ch/slsdesc/optic/magnets.html.

The ElementType in OPAglobal contains a pointer to a linked list of NameListType, which contains real names of elements, name of power supply, i.e. control channel, polarity etc.

lteconvert, madconvert and madseqconvert try to make Elegant *.lte and MAD *.mad and *.seq files understandable for OPA. They work on the TextBuffer string by exchanging keywords and inserting conversion factors without analyzing the lattice itself. This is then done by a subsequent call to LatRead. Tracy *.lat and Elegant *.ele files can be digested directly by LatRead.

WriteLattice accepts a variable mode of value 0..5 to write an OPA, Tracy-2, Elegant, MAD-X, BMAD or Tracy-3 file. mode 6 writes an OPA-file with all arithmetic expressions expanded into numbers.

Different beam dynamics codes use different units (e.g. degrees or radians) and different definitions of magnet order and strength. Only OPA, Elegant and Tracy-3 handle correctly nested inversions of segments to implement an asymmetric element (e.g. dipole with different entry and exit edge angles) correctly in the lattice. Export to the other programs requires to create and insert inverted segments (prefix I_) for all segments containing asymmetric elements.

All programs mentioned above allow arithmetic expressions instead of plain values for element parameters. However, unlike the others, Elegant uses inverse Polish notation, which is not included here, therefore expressions are expanded, i.e. resolved to values, for Elegant.

Further export of data is straightforward: first variables, then elements, then segments. The

complete lattice file is returned as one long string.

Type TextBuffer is a linked list of characters and has historical origin. It could be replaced by a long string, which would be much simpler.

4.2 Text Editor

texteditor

A simple Notepad-like text editor to edit the lattice file. The content of the edit window is one variable of type textbuffer, i.e. the editor "does not know" what the content means. At exit or when a test is requested, opalatticefiles>LatRead is used to analyze the lattice.

Uses: OPAglobal, opalatticefiles

Public procedures

Init called from opamenu accepts a handle to the drop down list of segments in order to change its content when segments are created or deleted.

LoadLattice called from opamenu calls opalatticefiles>WriteLattice to write the lattice file into the text window EdtWin. Then the text window is copied into a text buffer to save its content.

Actions

(text window) → disableOKBut disables the **OK** button when text is changed. No other action is done on text input.

Test → filetest performs a test of the input by calling opalatticefiles>LatRead. Before, the input is copied into a buffer, then, by calling OPAglobal>PassErrLogHandle, the output from LatRead is redirected from the opamenu main log window to the local error log window myerrlog, because the main GUI may be covered by the text editor GUI. If the test is successful, the **OK** button is enabled. Then the handle for output is set back to the main GUI.

OK → fileget reads the lattice in the same way, now knowing that the input is a valid lattice, and exits. At Exit, the segment drop down list in the mnain GUI is set to the last segment, from which the lattice was built by opalatticefiles>LatRead.

Cancel → fileRestore ignores the text window, reads the lattice from the buffer as saved at Init and exits.

Nowadays better text editors may be available, which include colored mark-up, auto-complete options and real time checks.

4.3 Interactive Editor

OPAEditor

The "OPA Editor" allows to create elements and compose a lattice without knowing element type names and lattice file syntax. It is more suitable for beginners, while the text editor may be more convenient for experienced users.

The GUI displays two lists, one for variables and elements, the other one for segments. In addition some global parameters like beam energy and default magnet aperture may be set.

Uses: `OPAglobal`, `EdElCreate`, `EdElSet`, `EdSgSet`, `../com/asaux`

Public procedures

`Init` called from `opamenu` accepts a handle to the drop down list of segments in order to change its content when segments are created or deleted. Init is also called internally to update the list contents.

Actions

(Element list) → `ListBoxEClick`: if the first line of the list is licked, the object `EditElemCreate` of class `EdElCreate` is initialized by passing handles to this list and to the `EdSgSet` object for editing segments. If another line of the list is clicked then object `EditElemSet` of class `EdElSet` is intitalized by calling two different procedures depending on if the clicked item represents a variable or an element, and a handle to this list is passed.

(Segment List) → `ListBoxSClick` initializes object `EditSegSet` of class `EdSgSet` by passing a handle to this list.

Beam energy → `EditGloEKeyPress`, (aperture fields) → `EditA*KeyPress`,
Magnet pole radius → `EdrrefKeyPress` are edit fields to accept values for beam energy, for element default apertures, and for default magnet pole inscribed radius.

(Comment window) (no event) text window `MemCom` accepts a comment, which will be saved in the lattice file between the {`com...com`} tokens.

Invert all dipole polarities → `ButDipInv` inverts all dipole polarities.

Expand undulators → `ButExpUndClick` expands an undulator into a series of dipoles and drift spaces creating the elements and re-defining the undulator as segment. Usually undulator is a basic element, and the series of dipoles and drifts is executed internally and not visible to the user.

Set all apertures → `ButAllAperClick` sets all apertures to the values given in the `EditAx`, `EditAy` fields thus overwriting individual element settings.

Set all (not only if larger) (no event) if checkbox `ChkAll` is checked, apertures of all elements are set, otherwise only those which have a larger aperture than the values in the fields.

Invert rotations in inverted segments (no event): If checkbox `ChkGloRi` is ticked or not defines if a beam rotation is inverted (like an asymmetric elements) or not when a segment containing the rotations is inverted. A corresponding flag `glob.rot_inv` is set and saved in the

lattice file.

Exit → ButExit saves data, updates the segment drop down list in opamenu and closes the GUI.

4.4 Element Creator

EdElCreate

This small GUI pops up, when `new entry` is clicked in the OPAEditor elements and variables list. A new element or variable is created after selecting its type and giving it a name. Then the GUI is closed and the EdElSet GUI pops up.

Uses: OPAglobal, EdElSet, EdSgSet

Public procedures

Init accepts handles to the OPAEditor elements list and to the segment editor EdSgSet, because if an element is created or subsequently modified by EdElSet, the changes should appear in these two GUIs. Then the list of element types is filled with the types defined in OPAglobal.

Actions

(Kind) → ComETypeChange selects the type of element from the list.

(Name) → EdENameChange accepts the name of the element.

Create → ButCreClick tests the input, if a type was defined, if the new entry is a variable or an element, and adds it to the corresponding arrays. Then the GUI closes itself and launches EdElSet using one of the two initialization procedures for variables and elements. If it is an element, in case the segment editor EdSgSet is open, it is updated.

Cancel → ButCanClick closes the GUI doing nothing.

Bug: there is no check for duplicate entries, should be added.

4.5 Element Editor

EdElSet

Main component of the GUI is a table to edit all parameters of an element or a variable. Some fields may contain algebraic expressions to calculate parameters from variables. This procedure is called in four different ways, for elements and for variables, and from editor or from optics design. For (iron dominated) magnets, a possible pole-profile is plotted.

Uses: OPAglobal, opticplot, ../com/asfigure, ../com/asaux

Public procedures

InitE, InitEVar, InitO, InitOVar are four different initialization procedures, for elements and variables, and for calling from the interactive editor OPAEditor or from the linear optics design opticview. In edit mode, a handle to the element list in opamenu is passed, the element index is taken from the list, and the element to be edited is taken from the original element array Elem. In optics mode, the index is given explicity and adresses the Ella subset of elements in the lattice, and a handle to the plot window of opticview is passed to enable an update of the plot when the element is changed. For variables there is only one array Variable (The arrays are defined in OPAglobal).
Then two private procedures are called, which are common for edit and optics mode, to fill the table with data of the element or variable. For elements several lines are filled depending on the type of element, for variables it is only one line. Some element properties and variables may have a value *and* an arithmetic expression (like "A+B"), which is a string for variables and a pointer for elements. If the string is empty or the pointer is `nil`, the value is taken in edit mode. In optics mode, the calculated value from the expression is shown too ("2.5 = A+B") for elements. For variables the expression is shown in both modes.
If the element is a quadrupole or a bending magnet, the GUI is extended by a plot area of type ../com/asfigure to show its pole profile.

getJelem just returns the index of the selected element in order to enable opticview to kill this GUI if the user assigns the element to a knob.

Actions

Done → butOKClick and **Apply** → butApplyClick both read the table and update the element. The **Apply** button is only enabled in optics mode and does not close the GUI in order to view how the change affects the optics.
The private procedure SetElem for updating the element performs several steps: In edit mode, it reads the name of the element, checks if it already exists, and, if not, if this element is used in segments, and asks, if all these entries should be renamed too. Then, in both modes, the table is read and the values are assigned to the element depending on its type. This includes conversion from convenient input units to internal SI-units. The validity of data is checked, i.e. if values are valid numbers, and if expressions can be executed without error. If everything is ok, then, in edit mode, data are written to the selected element in the Elem array and updated in the elements list of the calling opamenu GUI. In optics mode, the element in the Ella array is updated, the optics is recalculated calling opticsplot>OpticReCalc, and the plot in the calling opticview GUI is updated by sending it a `repaint` event.
For variables, the private procedure SetVar performs similar actions: In edit mode it reads the name, checks if it already exists, and, if not, it this variable is used in any expression, and asks if all these entries should be renamed too. Then in both modes, its expression is read from the table. (If the expression can be converted into a valid number, this value is assigned and the expression set to an empty string to identify a primary variable, which is a pure number.) Then, depending on the mode, the opamenu list is updated or the opticview plot is updated.

Cancel → butCancelClick does nothing but closing the GUI.

(Parameter table) → TabDrawCell checks if input is a valid number and marks it red in case of error. This is not done in edit mode, since input may be an expression. In optics mode, expressions as input are not editable.

ppaint reacts on repaint events for the form and plots the magnet profile, it is activated at initialization.

4.6 Segment Editor

EdSgSet

The GUI displays an editable string grid with names of elements and segments, which are marked in color depending on its kind and type.

Uses: OPAglobal, ../com/asaux

Public procedures

Init passes a handle to the segment list in OPAEditor and loads the abstract element sequence of the segment, which was clicked in the list, into the string grid. If entries contain an inversion flag or a multiplication factors, this is written to the grid cell too. The segment name is written to the (Modify segment) name field at top, and its periodicity (replication factor) to the (periodicity) field at bottom. If the first line of the segment list was clicked, a new segment is to be created and grid and name edit field stay empty.

Actions

(Grid) → drawCell analyzes the content of the cell and assigns a color code for element type, segment or unknown using private procedure ElemCol_C.

(Grid) → gridKeyDown reacts to pressing the CTRL+INSERT or CTRL+DELETE keys on the keyboard by inserting or deleting a grid cell, and shifting up or down the higher cells using private procedure ShiftCells.

(Modify segment) → EditSegNameChange: If the name of the segment was changed, the button to create a new segment is enabled.

update → butokClick and **create** → butcreClick call private procedure SaveSeg to read the grid cells and edit fields, and to perform several checks before creating or updating the segment:
If the grid is empty and the segment is not used by any other segment, then the segment is deleted, otherwise an error message is sent.
If the name is invalid or if the name already exists, or if the segment contains unknown names, error messages are sent.
If tests were successful, in case of (create) a new segment is appended to the list, in case of

(update) the existing segment is deleted first by disposing all its abstract element pointers in OPAglobal>ClearSeg.
If an existing segment was renamed, the entry is updated in all others segments. Then, the new segment sequence is saved as linked list of abstract elements by OPAglobal>AppendAE. If the segment was deleted, it is removed from the global Segm array and from the OPAEditor segment list, else the list is updated. Finally the exit procedure destroys the GUI.

delete → butdelClick sets all grid cells to empty strings and performs the delete procedures for an empty segment as described above, and exits.

cancel → butcanClick does nothing but exit.

5 Linear Optics

5.1 Linear optics design

opticview

This is the main GUI for linear optics development. It contains a plot window to show optical functions along the lattice, a table to display beam parameter results, a couple of knobs to be assigned to elements and several buttons for various options and for changing the display. Unlike all other GUIs, the plot window here is not based on ../com/asfigure, because it is too complex. All the beam dynamics calculations are done by opticplot.

Uses: knobframe, Opticstart, OpticTune, OpticWOMK, OpticEnvel, OpticMatch, opticplot, OPAtune, EdElSet, OPAglobal, mathlib, ../com/Vgraph, ../com/asaux

Public procedures

Init is called from opamenu passing a handle to a tune diagram as defined in OPAtune, which is further passed to opticplot. A Vplot object as defined in ../com/Vgraph is created, the corresponding variable for the plot window named vp, however, is defined in opticplot, where the calculations take place. A handle to the parameter table tab is also passed to opticplot. Then some inital values are set for various data.
If the lattice contains variables, copies of their values are saved, and all elements containing expressions are evaluated to check if the variables are used or not in the lattice. For each variable in use, a label is created underneath the plot window. If it is a primary variable (i.e. a number), which can be edited, the label is yellow, if it is a dependant variable (i.e. an expression) the label is pale yellow.
Finally the GUI is resized based on saved user settings. This causes a repaint event of the plot area:

Actions

(GUI resize) → FormResize reacts to a resize of the GUI by the user or by the Init procedure. The GUI is resized by reserving space for buttons etc. and adjusting plot area, parameter table and labels for the variables to left over space. Then as many knobs as fit into the GUI are created. these knobs of class TKnob are defined in Since a knob may affect the values of other knobs (in case it will be assigned to a variable), each knob receives handles to all other knob calling public procedure BrotherHandles of knobframe. If the GUI was shrinked, knobs which don't fit in anymore are freed (i.e. destroyed). Finally, the sizes of all components of the GUI are calculated and set.

(plot area) → pwpaint is not activated by the user but by the system if the paintbox of the plot area receives a generic repaint event. This is also done by opamenu after calling Init. The event calls the private procedure MakePlot, which performs these actions:
The plot is initialized, i.e. plotting axes etc. The abscissa is always the longitudinal coordinate along the lattice. Regions representing magnets in the lattice are calculated, plotted and saved to later enable reaction on mouse operations.
If MakePlot is called for the first time, the start GUI Opticstart is launched in front of this GUI and will start the first calculation calling opticplot>OpticCalc. Finally the resulting optical functions are plotted.
This way to launch the start menu is a bit awkward, there may be a better solution.

(plot area) → pwMouseMove shows the name of an element when hovering over.

(plot area) → pwMouseDown, pwMouseUp drag an element to a knob, if the left mouse button was pressed inside an element region and released inside a knob region. If an element editor EdElSet is open for this element, it is killed. If the element is already connected to another knob, an error message is displayed.
If the element is double-clicked, the element editor EdElSet is opened in optics mode, and if there was a connection to a knob, it is released.
If the right mouse button is pressed, the optical function values at this location are shown in the lower part of the table tab.

(variable labels) → varbutMouseDown, varbutMouseUp perform the corresponding actions to a variable, if the mouse is pressed inside the region of a yellow variable label and released inside a knob region.

Start → butStarClick launches the start GUI Opticstart again, to change initial parameters of the calculation.

PlotMode → butenvlClick launches the GUI OpticEnvel to select plot of beta functions, envelopes or magnetic fields, and to set some plot parameters.

TuneMatrix → buttuneClick launches the GUI OpticTune to smoothly adjust the lattice tune within a small range by changing all quadrupoles.

Matching → butmatClick launches the GUI OpticMatch for automatic matching of optical functions to target values by changing selected knobs (i.e. magnet strengths). Since some

magnets may be connected to knobs, a handle to all knobs is passed that the matching program may set them to new values.

Write OMK → butwomkClick launches a small GUI OpticWOMK to select the optics markers to overwrite with the current optics data. (An optics marker is an element to store local optics data, which may be used as a starting point for forward/backward optics calculations.)

linear → butnlinClick toggles between linear and non-linear calculation by setting the flag UseSext, and performs the calculation. The caption of the button is changed to **nonlinear** in non-linear mode. This option has only an effect on off-axis beams.

Kicker OFF → butkickClick toggles between kickers on and off by setting the flag UsePulsed, and performs the calculation. The caption of the button is changed to **Kicker ON** if kickers are on.

->txt → butdataClick writes four text files to the working folder, which are named (opa file name)_data, _beta, _mag, _rad.txt. They contain the equilibrium beam parameters, the beta functions along the lattice, the magnet field data and another output of beta functions for radiation calculations.

->EPS → butepsClick exports the plot to an encapsulated postscript file in the workig folder, which is named (opa file name)_betas, _envel, _magfd.eps depending if betas, envelopes or magnetic fields are shown. The procedure PS_start and PS_stop are called to switch on and off the postscript mode in ../com/Vgraph.

Exit → butexitClick calls OPAglobal>EllaSave to check for changes of element parameters and to ask the user to save the changes. Then user settings are saved and the GUI is closed.

(GUI close) → FormClose closes the GUI without saving anything.

<–> → buzoominClick zooms into the plot. The six other similar buttons perform similar actions like zoom out, shift left etc.
Internally, elements are subdivided in slices of a maximum length corresponding to one pixel on the screen, so a mini-beta-focus or the dispersion oscillation inside an undular may be resolved!

∧**B(x)** → buyupClick magnifies vertically the plot of betafunctions or the horizontal envelope, if in envelope mode. In magnetic field plot mode, it has no effect. The four other similar buttons are to shrink the beta plot and to magnify/shrink the dispersion plot, and to return to automatic scaling.

save → bucsaveClick saves the plot data to a second set of curves, which then is plotted in a darker color then the current curves in order to have a visual comparison of the changes. These actions take place in opticplot.

clear → bucclearClick removes the saved curves.

5.2 Parameter knob

knobframe

This component provides a knob to control an element parameter considered as strength as defined by `OPAgloal`>`putkval/getkval`. Values may be set by slider or edit field, including range limits and a reset function. As many knobs as fit into the GUIs are dynamically embedded in optics design `opticview` and orbit correction `OPAorbit`. Knob actions trigger calculations and plots. Knobs may stay passive and only display the value, if the parameter is an expression controlled by a variable (which may be connected to another knob). Therefore a knob has to know his fellow knobs and to trigger their updates.

Uses: `opticplot`, `OPAglobal`, `mathlib`, `../com/asaux`

Public procedures

`Init` gives a name using an index provided by the calling GUI (because it numbers the knobs) and initializes the knob as being not yet connected.

`SetSize` adjusts the knob to the size as given by the calling GUI. There is a minimum and maximum size, within this range the knob component is "elastic". If the range is exceeded, the number of knobs is adjusted by the parent GUI.

`Brotherhandles` accepts handles to the other knobs in the GUI.

`Load` and `LoadVar` connect the knob to an element or variable, accepting as input the index of the element or variable and a handle to the plot window of the parent GUI. If the parameter is an expression, the knob stays in passive mode, if it is a number, the knob is active. Element/variable name is written to the knob and range limits are set to values of same magnitude like the current value. The current value is saved. The element/variable is tagged in order to not connect it twice.

`UnLoad` removes the element/variable tags and sets back all captions to the initial status of being not connected.

`KUpdate` sets the knob to a new value by adjusting edit field and slider, and, if exceeded, also the range limits. If the knob was updated manually, private procedure `Action` is called to trigger calculations and plots and, if needed, to update the other knobs. If the knob is updated in this way from another plot, no further action takes place.
If the knbos are components of `opticview` the *optics* is re-calculated and the plot is updated depending which mode (betas, envelopes, magnetic field) is set. If the knob is component of `OPAorbit` the *orbit* is re-calculated and the plot is updated.
If the knob is connected to a variable, changing it may affect other knobs connected to elements or variables using *this* variable in an expression. So all elements' expressions are evaluated to update their values, and, if they are connected to another knob, the `KUpdate` procedure is called for the *other* knob (therefore it's not a circular reference). Of course, this affects only knobs which are in passive mode. This feature is yet only implemented for optics mode, since in orbit/injection mode no expressions are allowed for corrector magnets and kickers, which are the only elements to be connected. Thus it wasn't needed, however this is an uneccessary restriction.

`getella` and `getvar` return the index of the connected element/variable.

Actions

(value field) → editKKeyPress calls KUpdate to perform actions and adjust the range if the input value exceeds it.

(slider) → sliderScroll only performs actions, since the slider cannot exceed the given range.

(min/max fields) → editmin/maxKeyPress as well as >< → butwidClick and <> → butnarClick adjust the range limits by setting minimum/maximum or widening/narrowing the range, all calling private procedure SetKrange, which sets the small range fields and the slider parameters.

Reset → butresClick resets the knob to the value it had when connecting it and performs the actions.

X → butfreeClick disconnects the knob calling Unload.

5.3 Start parameters

Opticstart

This GUI pops up on initialization of the optics module opticview or the orbit/injection module OPAorbit (see sec.8), or when pressing the **Start** buttons in these modules. The starting conditions for the optics calculation are to be selected, either periodic/symmetric, or forward/backward from start/end of lattice or from one of the optics markers. In case of forward/backward calculation the initial beam parameters may be entered manually, in orbit/injecition mode this is only the orbit. In optics mode it includes also the normal mode beta functions, the dispersions and the elements of the coupling matrix.

Uses: opticplot, OPAglobal, mathlib, ../com/asaux

Public procedures

Load accepts a number for the mode of operation, which is 0,1,2 for optics, orbit, injection, and a handle to the parent form, which is either opticview or OPAorbit. Depending on the mode radio buttons to select periodic/symmetric or forward/backward options are enabled or disabled (per/symm makes no sense in injection mode). Then the lattice is searched for optics marker and corresponding radio buttons are added to the GUI. In optics mode a check box to perform calculations with or without coupling becomes visible and is checked at start if the lattice contains coupling elements. Finally the panels to enter/edit initial beam parameters are enabled/disabled and filled with data calling the private procedure set_pan_ini.

Exit is called from the parent GUI (opticview or OPAorbit) on exit to close this GUI too if it is still open.

Actions

(radio buttons) → rbutClick enables the tables to edit initial parameters depending on the selection of periodic/symmetric or forward/backward calculation, calling the private procedure set_pan_ini, and enables the buttons to start the calculations.

(periodic solution) → rbutperChange, this radio button has an additional event to hide/unhide the **flip** check box to start with flipped solution in case of coupling. (Further a button **pp** appears, which is a temporary test only.)

coupling → chk_coupChange hides/unhides the **flip** check box.

dp/p[%]= → but_dppClick unhides an edit field to enter a value for momentum offset, and a check box to select if one calculation for $Deltap/p$ or three calculations for $0, \pm\Delta p/p$ are to be done.

Apply → butapplyClick (or **Close** → butcloClick) starts the calculation calling private procedure Go (and closes this GUI). Before starting the calculation several flages are set depending on the mode (optics with or without coupling, orbit, injection) and the initial conditions. The procedures OpticCalc>or OrbitCalc>and, afterwards, the corresponding plot procedures are called from opaplot.

Exit → butexitClick closes this GUI and the parent GUI (opticview or OPAorbit).

(Coupling panel) → ed_couKeyPress,ed_couExit check the input into the edit fields of the coupling matrix (rarely used).

5.4 Optics plot mode selection

OpticEnvel

This GUI is launched by opticview to select what to show: beta functions (normal mode and/or projected) and dispersions, envelopes with orbit and apertures or magnetic fields. Additional parameters may be set.

Uses: OPAglobal, mathlib, ../com/asaux

Public procedures

Load sets GUI components (check boxes, fields etc.) for the current plot settings (explained below) and saves initial settings in order to restore them later.

Actions

(Beta, Envelope, Mag. field radio buttons) → rmodClick selects one of the three plot modes (Betas, Envelopes, Magnetic fields), unhides the corresponding panel and hides the two other, and enables options depending on flags: one flag is coupling, another flag is for using equilibrium emittances or input emittances. Further the data

table on the right side of opticview is configured, and table and plot are updated calling procedures opticplot>InitBetaTab,FillBetaTab. Then the private procedure MakePlot>sets plot ranges etc. and calls the corresponding plot procedures opticplot>PlotBeta,EnvPlot,MagPLot.

Apply → butgoClick also calls for a new plot and updates the data table.

Close → butcanClick closes the GUI.

Beta and Dispersion panel:

Fix plot range → cbxbetamaxClick enables the fields for max. beta and dispersion if checked. Then, **Apply** will read the edit fields **max. betafunction** and **max.dispersion** and fix the plot range. Otherwise the plot wll auto-scale.

normalmode → chk_betabChange and **projected** → chk_betxyChange select to show normal mode and/or projected beta functions. Without coupling this is the same. (The third option **n.mode one-turn** is only a test.)

Dispersion → rbu_dspChange, **det(C)** → rbu_cdetChange, **Orbit** → rbu_orbiChange radio buttons select to show the dispersion functions, the determinant of the coupling matrix or the orbit.

Envelopes panel:

use equilibrium / use input values → rbuClick radio buttons select to use equilibrium emittances and energy spread or manual input values for envelope calculation. In the equilibrium case, if the lattice is uncoupled, the field **emittance coupling** is enabled, otherwise both emittances from the coupled calculation are used. In the input case, three fields **horizontal emittance**, **vertical emittance** and **rms energy spread** are enabled.

Magnet field panel:

Fix plot range → chk_bfieldfixClick enables the fields for min. and max. field if checked. Otherwise the plot wll auto-scale. **Apply** will read the **Reference radius** field for pole-tip field calculation, and the two edit fields **min. B[T]** and **max.B[T]** to set the plot range.

5.5 Tune matching

OpticTune

This small GUI launched by opticview calculates a 2×2 sensitivity matrix how the lattice tunes depend on a scaling factor applied to the two groups of all horizontally and all vertically focusing quads. A new tune may be entered manually, and the inverse of the sensitivity matrix is then used to adjust all quads correspondingly. This procedure converges for small tune changes and was also used in SLS control by exporting the sensitivity matrix to EPICS channels from module OPACurrents. The code is simple and may not need further explanation.

Uses: opticplot, OPAglobal, mathlib, ../com/asaux

5.6 Optics marker update

OpticWOMK

Optics markers are markers in the lattice which can store beam data (betas etc.) and may be used as starting points for forward/backward calculations by Opticstart. This GUI establishes a list of all optics markers in the lattice with tick marks and asks if their data should be overwritten with the current optics solution. This may be ambiguos since an optics marker may appear several times in a lattice, so the data refer to the first instance, counting from begin of lattice. The code is simple and may not need further explanation.

Uses: OPAglobal, ../com/asaux

5.7 Lattice calculations

opticplot

Linear beam optics includes closed orbit finder, periodic solution for coupled and uncoupled lattices, and radiation integrals. This unit has handles to the plot areas and tables of the optics and orbit/injection GUIs, opticview and OPAorbit. Plot routines show beta functions and dispersions, or orbit, envelopes and apertures, or magnetic fields. Lattice and local beam parameters are filled into the tables, and several output files can be printed.

Uses: OPAElements, OPAtune, OPAglobal, mathlib, ../com/Vgraph, ../com/asaux

Public constants, types and variables

Several variables used by opticview, Opticstart etc. are defined in the interface section, i.e. as public, for convenience (i.e. lazyness). Most of them are flags to control the flow. Few of them may require an explanation:

vp defines a Vgraph object for the optics GUI opticview, which creates it at start. Later, opticplot itself will plot to vp. The orbit/injection GUI OPAorbit, however, defines its own plot areas of class asfigure and only gets the data from opticplot. The reason for this different solutions was partly historical, partly due to the higher complexity of the opticview plot.

Opval is a dynamic 2-dimensional array of OpvalType to save optics parameters after each lattice element and for up to three calculations, e.g. for on- and ±off-momentum.

Obeam is a 3-element array to save beam parameters like tunes, emittances etc. for up to three calculations.

Public procedures

allocOpval allocates the array of optics values.

shiftOpval shifts up the optics values for one solution the next row in the lattice; this is used to save a solution for comparison with a newer one.

setTabHandle receives a handle to the table in opticview in order to fill in optics results

setTunePlotHandle receives a handle to the tune diagram GUI OPAtune in order to plot working points.

setOrbitHandle receives a handle to the OPAorbit GUI. Actually it is only needed to send it a repaint event, it will then get and plot the data from opticsplot.

Beam dynamics procedures:

OptInit initializes all variables (beta functions, transfer matrix etc.) to be used for the optics calculations.

Lattel is the basic procedure for all calculations. It propagates the optical functions through one lattice element. Input is the lattice index and a mode flag telling it what to include in the calculation (integrals, misaligmnents etc).

ClosedOrbit tries to find the fixpoint of the one-turn map, i.e. the closed orbit, by a Newton-Raphson root finder with the *local* transfermatrix as Jacobian. It returns a flag to tell if it failed or not. For on-momentum calculation without misalignments, it is trivial and the orbit is on-axis. Else down-feeds from non-linearities affect the local transfermatrix and require some iterations, usually only a few since the method converges quadratically.

Periodic tries to find the periodic optics solution, if the closed orbit was found. Depending on coupling it switches to two procedures:

Flatperiodic calculates the periodic solution for uncoupled, i.e. flat lattices without coupling elements, where horizontal and vertical dimension can be calculated independently, and vertical dispersion is always zero. The algorithm is described in Klaus Wille's book [7].

NormalMode calculates the periodic solution for coupled lattices using the Edward-Teng formalism in the version of Sagan and Rubin and implemented as described in [6]. Section 2 of the "inside OPA" guide [3] gives a summary. If successful the procedure returns the normal mode beta functions, which have no physical meaning and exist in an uncoupled system (coordinates a, b) and the coupling matrix, which translates the normal mode system to the real coordinate system (coordinates x, y). It further returns the 4-vector of dispersions and the lattice tunes.

Symmetric calculates the symmetric solution with $\alpha_x = \alpha_y = D'_x = 0$ and works only for uncoupled lattices. It is rarely used and not well tested. The algorithm is also described in [7].

LinOp is a *private* procedure to propagate the beta functions through the lattice and save the values after each element to the Opval array by calling procedure StoVar. Usually it would start at the begin of the lattice with initial conditions from the periodic solution or entered manually. But it may also start at an optics marker and propagate forward and backward to end and start of lattice. Care is taken to handle inverted elements and derivatives of optical parameter and

to sum up integrals correctly. Depending on the input flag `latmode` only the beta functions and dispersions are calculated or chromaticities and radiation integrals are calculated too by passing the flag to the `Lattel` routine, which passes it further to the element propagation routines.

`OpticCalc` performs the full linear optics calculation by calling `ClosedOrbit`, `Periodic` and `LinOp`. Afterwards, the working point is plotted to the tune diagram (in case of periodic solution only) and the table in `optiview` is filled calling procedures `FillBeamTab` and `FillBetaTab`.
The code is a bit messy due to the different cases it has to handle – periodic/symmetric solution or forward/backward from an optics marker, how many calculations to do, on- or off-momentum.

`OrbitCalc` is a simplified version of `OpticCalc` used by `OPAorbit` for calculation of orbit only. An additional feature is the ability to do several turns which is useful in injection simulation.

`MatchValues` is a function returning an array of type `OPAglobal`:`MatchFuncType` containing optics values at the begin and end of the lattice, and optionally, at an intermediate point. It is used by `OpticMatch` for matching optics parameters to target values.

Plot procedures:

`SliceCalcN` calculates optical functions *inside* an element by subdividing it into slices of a length corresponding to one pixel on the screen. Calculations start at each lattice element, where optical functions have been before calculated by `Linop` and saved in the `Opval` array. Elements outside the plot window are skipped, if the user zoomed in to a part of the lattice in `opticview`. The variable `OPAglobal`:`CurvePlot` stores the start and end point and the number of slices and their length, and the optical functions at the slices are saved in the variable `OPAglobal`:`Curve`, which is a linked list of pointers for storing optical data. Procedure `OPAElements`:`SlicingN` performs these calculation and generates the curve.
It's a unique feature of OPA to display the optical functions at pixel resolution. This allows to export nice plots ready for publication.

`PlotBeta` displays the beta functions and dispersions. The procedure is lengthy due to the various plot modes as selected in `OpticEnvel`, but in principle it is relatively simple: After setting the ranges, the internal procedure `BetaCurveN` is called to generate the curves calling `SliceCalcN` and to plot them. More than one curve for each function is plotted if a previous curve was saved for comparison, or if a $0, \pm\Delta p/p$ calculations was done.

`PlotEnv` displays the envelope functions. Before, they need to be calculated. Therefore several arrays are created to hold all the data for the orbit and the different contributions to the beam size (from dispersion, from coupling). They are calculated either using input or equilibrium emittances and energy spread as selected in `OpticEnvel`. Also the apertures are plotted calling procedure `PlotApertures`.

`PlotMag` displays the pole-tip field of the magnets. No slicing is done since the magnets have no longitudinal variation of the fields (longitudinal gradient bends are approximated by a stack of dipoles). In case of the solenoid, the longitudinal field is shown. For combined function magnets only dipole and quadrupole field is used for pole-tip field calculation. The reference radius as set in `OpticEnvel` is assumed to be the same for all magnets.

`PlotOrbit` is used by the orbit & injection GUI `OPAorbit` only, whereas the other plot

procedures are only used by the optics GUI opticview. OPAorbit does the plotting by itself, so this plot procedure just sends a repaint event to its formhandle.

Output

FillBeamTab and FillBetaTab write equilibrium beam parameters and beta functions at the location which was right-clicked (see above: opticview action (plot area) → pwMouseDown) to the table on the right side of opticview. In envelope mode, beam sizes, correlations and beam tilt angles are calculated and displayed.

Print... These procedures print various data to files: lattice data in text or html format, beta functoins curves, magnet and radiation data.

5.8 Element calculations

OPAElements

This unit contains the procedures to propagate optical parameters through the different elements in the lattice. They all receive the variable mode telling them what to include in the calculation, by evaluating OPAglobal>switch. Most elements receive the varialbe idir to set forward or reverse orientation of the element (for example important for a dipole with different edge angles). All other parameters are explained in the user guide [2]. The flags UseSext,UsePulsed in OPAglobal switch on/off non-linearities and kickers. Coupled calculation is selected by the element procedures. Another set of procedures performs the sliced calculations for thick elements to provide data in pixel-resolution for a nice display.

Uses: OPAglobal, mathlib

Procedures

This section tries to list the procedures in a logical order (not as they apppear in the file) for best understanding. Some are public, to be called by opticplot, others are private and commented out in the INTERFACE section of the file.

Beam propagation

Propagate receives the transfermatrix of an element and a flag on coupling to propagate the optical functions from start to end of the element using two procedures:

MCC_prop proceeds *with* coupling to propagate the normal mode beta functions and the coupling matrix. In the beginning, on- and off-diagonal 2×2 submatrices are extracted from the 5×5 transfer matrix, and a test is done, if a mode flip will happen. Then follows a longer section to handle a possible mode flip. This section is partly experimental and needs further elaboration. It does not work correctly with regard to tunes and integrals in all cases. At the end follows the normal case which works reliably with weak coupling. Phase advances are calculated by procedure PhaseAdvance and added to the ring tunes, and the dispersion vector is propagated,

which always works.

MBD_prop proceeds *without* coupling where the transfer matrix is known to be block-diagonal. Nevertheless, possible coupling in the beam has to be propagated too by transforming the coupling matrix, which may have non-zero elements in this case.

PropForward just copies optics parameters after propagation (suffix 2) to parameters before propagation (suffix 1) in order to continue with the next element.

PhaseAdvance calculates the block-diagonal normalized normal mode matrix from the local beta functions using the transformation matrix from function CirMat, and extracts the phase advances from the traces of the sub-matrices. getdTune is a simplified version for a transfer matrix which is known to be block-diagonal.

GetBeta12 calculates the projections of the normal mode beta functions to the physical beta functions.

Pathlength calculates the path length by estimating the second order matrix elements as defined in TRANSPORT (see comment in code for details).

The elements

The ..._Matrix functions return 5×5 transfer matrices of the basic linear elements. Since OPA does not include longitudinal dynamics, there are no rows and columns for phase/time and the 55-element is always 1. The fifth column of the matrix contains the dispersion production vector. Drift space, quadrupole, sector magnet and edge kick have block-diagonal matrices, i.e. they don't couple. Coupling elements are the explicit rotation and the solenoid. The Bending magnet matrix is a sector matrix sandwiched between edge kicks. The matrix functions are called by the element procedures. OPA knows 17 different elements as listed at the very beginning of `OPAglobal`:

There are no procedures for markers, optics markers and photon beam markers, because they don't act on beam dynamics. There is a placeholder procedure Monitor for beam position monitors but it does nothing. The septum element –as seen from the stored beam – is handled as a drift space in beam dynamics.

DriftSpace, Quadrupole and Bending are the standard linear lattice elements. Actually Bending switches to Quadrupole if the bend angle is zero, and Quadrupole switches to Driftspace if the gradient is zero. So Bending is the most complex procedure and should be explained in some detail. It performs the following steps: at first the magnet is misaliged, if misalignments were requested, then a rotation is applied if the magnet is tilted. Depending on the direction of the magnet, the edge angles are interchanged and the edge kick matrices are calculated. Then the beam (i.e. the optical functions) is propagated through the entry edge because the values at the beginning of the sector are needed to estimate the phase advance inside the sector (procedure getdtune) in order to set the right number of points for numerical evaluation of the radiation integrals. Before doing so, the complete bending magnet matrix including entry edge, sector and exit edge is calculated twice, first on-axis (TM0) as needed for the closed orbit finder, then off-axis (TM) to include dispersion downfeeds for the integrals. If radiation integrals or chromaticities are requested, the sector is split in a number of slices, the

beam is propagated step-wise, and the integrals are obtained by Simpson's rule (see section 3 in [3] for details, also for the other elements). Finally, the beam is propagated through the complete bending magnet, and rotation and misalignments are undone at exit. `Quadrupole` contains a simplified version of this algorithm, but including radiation integrals too, since there may be off-axis contributions.

`Combined` is a bending magnet which also has a sextupole component. It is modelled as a series of bending magnets with thin sextupole kicks between. Having both `Combined` and `Bending` has historical reasons, and maintaining both is for backward compatibility, but in principole they could be united.

`Undulator` contains a loop over rectangular bends and drift spaces, where the parameters of these sub-elements are set by the procedure `UnduFacs`. For the radiation integrals a simplified algorithm using averages is applied since the bends inside the array are small.

`Solenoid` is a simplified treatment of the solenoid providing beam rotation and focusing in both planes. Note, that the transverse vector potential components of the solenoid are not taken into account, also contributions to chromaticity and radiation are not considered.

`Rotation` is an explicit beam rotation. It's rarely used, because rotated elements like skew quadrupoles are rather set up by passing a rotation angle to the corresponding element procedure. Then the element matrix internally is sandwiched between the rotation and its inverse.

`ThinSextupole` and `Multipole` are thin multipole kicks – basically, a sextupole is just a multipole of order 3, but the dedicated procedure executes faster. The procedures apply a non-linear kick to the orbit and a transfer matrix to the optics for off-axis gradient downfeed. (For order 2, quadrupole, there is a gradient on-axis too.) Contributions to radiation integrals are usually negligible and therefore not included. The `Multipole` procedure uses explicit expressions up to octupole (order $n = 4$). For higher orders the general complex expression is evaluated (see comment in code). A vertical offset of the beam requires coupled treatment due to skew quad down-feed.

`Sextupole` is a thick sextupole, internally modelled as a series of drifts and thin sextupole kicks as the most simple second order symplectic integrator. A corresponding procedure for a thick general multipole does not exist yet.

`Kicker` is a time-dependent thick multipole. Parameters delay and period (of half sine bump) and the time of flight from the start of the lattice define the kick strength applied. Like the thick sextupole, the kicker is subdivided in a series of drifts and thin kicks. The transverse shape of the kick can be selected to be of multipole or sine shape (BESSY type NLK). Note that for the NLK, the kicks are only correct in the midplane ($y = 0$).

`HCorr` and `VCorr` are correctors, i.e. weak dipoles which just give a kick to the orbit and to the dispersion function without further effect on the beam optics.

Slicing

The `SliceSet` function calculates the number of slices for the optical functions inside an element to be displayed in one-pixel resolution on the screen. The record `OPAglobal`:`CurvePlot` contains a field `slice` for the length of the slice in physical coordinates, i.e. in meters, calcu-

lated by opticview>Zoom. The SliceSet function checks if the element is outside, partially inside or completey inside the current plot window, or if the plot window is completely inside the element. It writes start and end position inside the element and the number and length of slices to the CurvePlot record and returns true if at least a part of the element will be visible in the plot area.

SlicingN receives transfer matrices msl for a slice, mof for the invisible part of the element left of the plot window, and min and mex for edge focusing at entry and exit of the element (applies to bends only). Additional matrices mof0,msl0 without dispersion down-feeds are supplied to show both orbit and dispersion functions correctly. The procedure further receives the optics values at the entry to the element, because these are known from the previous run of opticplot>Lattel. Then optical parameters at the slices are calculateed using a simplified propagation procedure and the results are appended to the OPAglobal:Curve variable for later plotting by opticplot. Finally the optics value at the end of the calculation (which may be still inside the element) overwrite the optics values from input.

SkippingN is only needed for undulators, which are modelled as a loop over drifts and bends, and where several of these basic elements may be left of the plot window. The optics values are propagated through these invisible basic elements to get the correct initial values at the first partially visible undulator pole. In this wasy it is possible to zoom into an undulator and see the dispersion oscillation!

The Slice(Drift,Quad,Sol,Bend)N procedures call SliceSet and calculate the required transfermatrices for SlicingN and SkippingN.

SliceUnduN loops over bends and drifts of the magnet array if at least a part of the undulator is inside the plot window.

SextKickN is a simplified version of ThinSextupole to propagate optics values through a sextupole kick.

SliceSextN and SliceCombN make use of SextKickN because they model thick sextupole and combined function magnet as loop over drifts or bends with sextupole kicks between.

5.9 Linar matching

OpticMatch

Matching implements a linear equation solver based on LU decomposition for the linear optics. For historical reasons, it does not use the solver in Mathlib>LUDCMP,LUBKSB but the algorithm was explicitly coded in the 1980's by Klaus Wille (procedure Step) , and is documented in his book [7]. Matching is done from a start point to an end point in the lattice. The lattice tunes and optics functions at an additional intermediate point may be included too. After selecting these options, a list of available knobs is established. Then the user selects the constraints, i.e. the optics parameters at end (and intermediate) point, sets the target values and selects at least the same number of knobs (e.g. quadrupole strength, variable etc.) from the list. The solver proceeds using the square matrix of most sensitive knobs, which is recalculated after each step

to take into account non-linearities. If successful, the method converges quadratically, if not it returns messages, what went wrong (procedure Term). Reduced step size can be set for slower but more robust convergence. It is possible to run a scan over some range of target values.
There are three panels at the right side of the GUI, one for the knobs, one for the matching process and one for parameter scans. Only one of them is visible at a time.
The algorithm itself with catches for errors as implemented over the years, is quite robust and could be used further in principle. However, nowadays better and well proven solvers may be available, which also include limit ranges for the knobs. Further, the present procedure is restricted to uncoupled lattices and to only one intermediate point besides start and end of matching, which sometimes is a nuisance. The present GUI is well conistent (how options are enabled etc.) but its rigid structure makes extensions cumbersome. It should be re-organized for example using tabs.
Expecting a rewrite of this module in future, only a brief summary of the most important actions will be given here.

Uses: opticplot, knobframe, OpticMatchScan, OPAglobal, ../com/asfigure, ../com/asaux

Public procedures

Load initializes the GUI. It receives handles to the PaintBox plot window and to the array of knobframe knobs in opticview in order to be able to set the knobs to new values and to show the new optical functions after matching. Then it searches for optis markers, which may be used as start, end or intermediate point for matching, and adds them to the ini/mat/midpoint pulldown lists. Several user settings are restored from a previous session, since often the same matching scenario is used again.

Actions

starting,matching,intermediate → (ini,mat,mid)pointChange call the private procedure knoblist to establishes the list of available knobs after selecting the start, end (and optionally intermediate) points for matching. For each knob a named check box is created.

Optics parameter check boxes → cselClick and **Knob check boxes** → knobSelect build the lists of constraints and knobs and set the **Go** button if there are at least as many knobs as constraints.

incl.bends → cbxIncBendsClick includes bending magnets in the knob list, because often they are not used, so the panel is less crowded if they are not shown.

Go → gobutClick hides the knob panelpanknobs and shows the matching panel panmat with a grid of knob values. Then the iteration starts until the procedure Step returns true because it terminated with success or failure, or the user stopped the iteration pressing **Abort** → bustopClick. The last option requires the Pascal procedure Application.ProcessMessages, which reacts to GUI events, to be called inside the loop. After termination buttons become active to show the result in opticview, to accept the new values, to reset to initial values or

to retry the calculation (for example with reduced step size to be entered in the **Fraction** [%] field, in case of bad convergence.)

Show → butShowClick actually only hides the OpticMatch GUI (which is modal!) to uncover the opticview GUI which is behind and shows the current optical functions anyway.

Accept → butaccClick saves the settings and closes OpticMatch.

Reset → butaccresClick restores the old knob values, recalculates the optics, saves the scenario and closes the GUI.

Retry → butrtyClick restores the old knob values and sets up for a new iteration.

Scan → ScanClick prepares for a parameter scan, i.e. a series of matching iterations for a range of target values. For this purpose, the scan panel panfig becomes visible, the modal GUI OpticMatchScan is launched to enter range parameters, and then the matching series are started, the results are saved and plotted eventually. The <, > buttons under the plot window allow to toggle between displays for the knobs and functions.

5.10 Parameter scan

OpticMatchScan

This very simple modal GUI is launched by **OpticMatch** → ScanClick that the user may enter the range for a parameter scan. At exit it writes the data to the variable opticplot:ScanPar, from where OpticMatch takes them to perform the scan.

Uses: opticplot, OPAglobal, ../com/asaux

5.11 Tune diagram

OPAtune

The tune diagram GUI is created at start of OPA already and provides the variable tuneplot, which is passed in the initialization procedures of the modules using it: opticview shows the working point, i.e. the tunes of a periodic solution. OPAmomentum shows the variation of tunes with momentum. OPAChroma shows an analytical estimate of the tune footprint. And OPAtrackP shows the amplitude dependent tune variation. In the background always shown is the web of resonance lines.

Uses: OPAglobal, ../com/asfigure, ../com/asaux

Public procedures

Diagram makes the tune diagram visible if a periodic solution exists. Input is the point in tune space where to center the diagram. The resonance lines inside the visible tune region up to the selected order are calculated by private procedure getLines and characterized as regular, skew,

non-systematic. The periodicity of the lattice is stored in `OPAglobal`:`NPer` and determines the selection of systematic resonances.

`AddTunePoint` adds a tune point to the diagram. If the input flag `connect` is `true` the point is connected to the previous point. The inputs `dpp, dppmax` are used to show off-momentum points in a color range. Used by `opticview` and `OPAmomentum`.

`AddChromLine` adds a line to show the chromaticity, i.e. the variation of tune with momentum, with chroomaticities up to third order and the momentum range as input. Used by `OPAChroma`.

`AddTushPoint` adds points from amplitude dependant tune shift tracking. Input `xmy` controls the color of the points: horizontal, coupled, vertical in blue, purple, red; outside aperture in grey. Used by `OPAtrackP`.

`Refresh` clears the diagram from points which had been added.

Actions

(On create) → `FormCreate` sets defaults and restores user settings.

(On resize) → `FormResize` adjusts the GUI if the user changes its size.

All other actions control the plot: The buttons (Order) + − increase/decrease the maximum resonance order. The **nsys** and **skew** check boxes switch on/off non-systematic and skew resonances. The buttons under the plot window are to zoom in/out, to shift the region right/left, up/down and to re-center. **EPS** exports the diagram.

6 Momentum Dependent Optics

6.1 Momentum dependence

OPAmomentum

6.2 Calculations

MomentumLib

7 Non-linear Optimization

7.1 Non-linear optimization

OPAChroma

7.2 Element controller

CSexLine

7.3 Result indicator

CHamLine

7.4 Result handles

ChromGUILib1

7.5 Element handles

ChromGUILib2

7.6 Vector diagram

OPAChromaSVector

7.7 Calculations

chromlib

8 Orbit and Injection

8.1 Orbit and injection

OPAorbit

9 Special Element Editors

9.1 RF bucket

`Bucket`

9.2 Longitudinal gradient bend

`LGBeditor`

9.3 LGB calculations

`LGBeditorLib`

10 Tracking

10.1 Phase space

`OPAtrackP`

10.2 Dynamic aperture

`OPAtrackDA`

10.3 Touschek lifetime

`OPAtrackT`

10.4 Calculations

`tracklib`

11 Layout, Currents etc

11.1 Machine Layout

OPAGeometry

11.2 Magnet currents

OPACurrents

getKfromI, getdKdIfac, getIfromK convert magnet strength in current and vice versa taking into account non-linear magnet saturation.

11.3 Test procedures

opatest

11.4 Editors

LGBeditor.pas/.lfm (19.1k)

---

uses OPAglobal, MathLib, ASfigure, ASaux

LGBeditorLib.pas (13.9k)

---

uses OPAglobal, ASfigure, ASaux

These two procedures optimize the field profile of a longitudinal gradient bend in order to minimize quantum excitation, i.e. the I_5 radiation integral. Magnet type, length, peak field and number of slices are set before starting a Powell minimizer. The first procedure is mainly the GUI and the minimizer, the second one contains physics and plotting.

11.5 Optics Design

OpticPlot.pas (98.3k)

---

uses OPAglobal, OPAElements, OPAtune, MathLib, ASfigure, VGraph, ASaux.

`OPAorbit.pas/.lfm` (73.0k)

---

uses `OPAglobal, OpticPlot, OpticStart, Knobframe, MathLib, ASfigure, ASaux.`

This unit calculates the beam orbit. It is used to either study orbit distortion and correction (usually in periodic mode) or to study injection (usually in forward mode) – a corresponding flag is set at start. Most procedureds in the unit are used to set up the rather complex GUI with knobs for correctors, kickers and BPMs and various panels for orbit correction, plotting or injection studies.

In orbit mode, correlated misalignments are applied, the response matrix is calculated and pseudo-inverted using an SVD procedure in order to correct the orbit. A loop function may run several error seeds to obtain some statistics.

In injection mode, kickers may be synchronized for correct timing, and the injected (and/or stored) beam trajectory is calculated for a few turns.

In both modes, when leaving the unit, results for misalignments and corrector settings, or for kicker settings, are saved internally and will be used in subsequent linear optics calculations or tracking.

There are problems with misalignments in tracking, not yet solved.

Note, that correctors and BPMs defined with reserved names `CH,CV,MON` in the lattice file are internally expanded to a set of individual elements, e.g. `CH001...` when unpacking the lattice `OPAGlobal>MakeLattice`.

11.6 Longitudinal optics

OPA does not contain true longitudinal dynamics, i.e. all calculations are for fixed momentum offset, and tracking proceeds only in 4-D, not in 6-D. There are no cavities and no acceleration.

`OPAmomentum.pas/.lfm` (26.8k)

---

uses `OPAglobal, OpticPlot, ASfigure, OPAtune, MathLib, ASaux.`

`MomentumLib.pas` (16.5k)

---

uses `OPAglobal, OpticPlot, ASfigure, OPAtune, MathLib, ASaux.`

These two procedures calculate linear optics (periodic or forward) for some momentum range, show the results and apply a polynomial fit. The first procedure mainly controls the GUI, the second one is for calculation and plotting. Results to fit and show are selected in the GUI. Results for path length are saved internally to be used when plotting the bucket (see next unit). Tune results are shown in a tune diagram.

A Powell minimizer was implemented once for a special purpose (non-linear bunch compressor study): it takes selected multipoles as knobs to adjust a momentum dependent function (e.g. $X(\delta)$) to a target function. However this implementation was "quick and dirty" and may be removed again.

`Bucket.pas/.lfm` (19.4k)

---

uses `OPAglobal, ASfigure, MathLib, Conrect, ASaux.`

The RF bucket is calculated based on up to five orders of momentum compaction and RF harmonics as described in Appendix **??**, `>CalcBucket`, and a contour plot of equipotentials and the separatrix is shown.

The coefficients for momentum compaction may be taken over from the previous unit on momentum dependent optics. The estimates for momentum acceptance and bunch length are saved for re-use in the Touschek tracking unit (see below).

11.7 Non-linear Optimization

Non-linear optimization uses a penalty function composed from a fixed set of Hamiltonian modes on one side, which are controlled by a variable set of sextupoles and octupoles on the other side. The dependencies are complicated, if the two minimizers are running (Powell for first and second order sextupole terms, SVD for first order octupole terms), or if chromaticity is automatically adjusted, or if target values for the (non-resonant) Hamiltonian modes are changed. This requires to pass handles from the GUI program `OPAChroma` to the frames for the Hamiltonian modes and the multipoles, `>CSexLine, CHamLIne` and between these. In order to avoid circular dependencies, two intermediate layers `>ChromGUILib1,2` were introduced.

The nested handles are rather messy. It works, but could be entangled and simplified.

`OPAChroma.pas/.lfm` (40.2k)

---

uses `OPAglobal, ChromLib, ChromGUILib1, ChromGUILib2, CHamLine, CSexLine, OPAChromaSVector, OPAtune, MathLib, ASfigure, ASaux.`

Initialization of the GUI, allocating the frames for Hamiltonians and multipoles and passing all the handles between, `>Start` and dimensioning of the GUI `>ResizeAll`. Set up (and start) the Powell minimizer for the sextupoles `>ButMinClick` and set up the SVD-minimizer for the octupoles `>ChkOctClick`.

`ChromLib.pas` (60.2k)

---

uses `OPAglobal, OpticPlot, OPAElements, OPAtune, MathLib, Vgraph, ASaux.`

The physics part: at start, all kicks from sextupole, octupole, combined function bend (and decapole) families are collected, and matrices are set up to get the Hamiltonian modes from the M_{sx} sextupole and M_{oc} octupole families. These are a $10 \times M_{\mathrm{sx}}$ matrix for first order sextupole, a $11 \times M_{\mathrm{sx}}^2$ matrix for the second order sextupole, and a $13 \times M_{\mathrm{oc}}$ for first order octupole. Matrix

elements are sums over optical functions at all kicks (thick sextupoles contain several kicks). Quadrupoles contribute to chromatic modes and are included as a constant offset vector. These calculations are rather time consuming but only done once at start `>ChromInit, S_Matrix`. Hamiltonian modes are calculated for the lattice structure considered as one period. In order to get the results for many periods, complex multiplication factors are required as described in Appendix **??** `>S_Period`.

Chromaticity up to third order is calculated from numerical differentiation `>ChromDiff`. Hamiltonian modes are calculated from multipole settings using the pre-calculated matrices and weight factors entered manually in order to visualize the results and combine them into a single, scalar penalty function, see Eq.**??** `>Driveterms`. Chromaticity is corrected with two selected sextupole families using a simple 2×2 matrix `>UpdateChromMatrix, GetLinChroma, ChromCorrect`.

For the octupole matrix a SVD decomposition is performed for correction of the second order sextupole terms which are first order octupole terms. The SVD weight vector is provided for filtering, since usually a "hard" correction using all weights does not work well `>Oct_SVDCMP`. Decapoles affect only the third order chromaticities. Other third order effects are not included.

`ChromGUILib1.pas` (1.9k)

---

uses `ChromLib, CHamLine.`

This small piece of code is mainly for passing handles to the`>CHamLine` frames.

`ChromGUILib2.pas` (2.7k)

---

uses `ChromLib`

Pass handles to the octupole SVD functions in order to enable automatic execution during minimization, and filters the SVD weight factors.

`CHamLine.pas/.lfm` (10.7k) [Frame]

---

uses `OPAglobal, ChromLib, ChromGUILib2, CSexLine, ASaux.`

25 of these frames are embedded in the GUI at start. They display the results for the 25 Hamiltionian modes and allow weight and target values to be set. A change of target or weight triggers an update of the calculations `>UpdateWeight, UpdateTarget`. Changing any multipole manually or by the minimizer of course changes these frames.

`CSexLine.pas/.lfm` (11.9k) [Frame]

---

uses `OPAglobal, ChromLib, ChromGUILib1, ChromGUILib2, OPAChromaSVector, ASaux.`

At start, one of these frames is embedded in the GUI for each family of sextupole, octupole,

decapole and combined function bend. It provides a knob for the magnet strength. Changing it triggers a calculation of the Hamiltonian modes. If the minimizer is running, the strength field has to follow `>UpdateVal...`

`OPAChromaSVector.pas/.lfm` (3.3k)

---

uses `OPAglobal, ChromLib, MathLib, ASfigure.`

Opens a small window to visualize the first order sextupole modes in the complex plane.

11.8 Tracking

Tracking is performed in unit `TrackLib`, used by the three GUIs for phase space, dynamic aperture and Touschek tracking:

`OPAtrackP.pas/.lfm` (44.8k)

---

uses `OPAglobal, OPAtune, TrackLib, MathLib, Vgraph, ASfigure, ASaux.`

Phase space tracking: the GUI contains four plot windows for the transverse phase spaces and Fourier spectra, and several panels for different options. Single particles are started from coordinates entered manually in edit fields or passed from `ASfigure` mouse events `>ButRunClick`. For amplitude dependent tune shifts (ADTS) particle start coordinates are stepped up to aperture limits `>ButTushClick`. For simulation of injection, a beam ellipse populated with many particles can be tracked `>StartEnsemble, ButBeamRunClick, TrackBeam`. Most of the unit contains event handlers and plot routines.

`OPAtrackDA.pas/.lfm` (30.9k)

---

uses `OPAglobal, OPAtune, TrackLib, MathLib, Vgraph, ASfigure, ASaux.`

Dynamic aperture tracking: particles are started on a grid covering the area (x, y), (x, δ) or (y, δ). The grid is successively refined to get an early impression `>gridSetup`. Physical apertures are calculate by projecting all apertures to the trackpoint `>Silhouette`. The other procedures are for event handling and plotting.

`OPAtrackT.pas/.lfm` (45.7k)

---

uses `OPAglobal, OPAtune, TrackLib, OpticPlot, MathLib, Vgraph, ASfigure,ASaux.`

Touschek tracking: Lifetime is calculated as described in Appendix **??** from bunch volume and momentum acceptance (MA). The MA is the minimum of the linear MA given by the beam pipe apertures `>CalcMALin`, of the RF-MA which is derived from input parameters or has been calculated previously by `Bucket`, and of the dynamic MA obtained from tracking and binary search for min./max. stable momentum offset `>MADyn, Trackdbins, TrackLat`.

The bunch volume is calculated from the periodic optics solution with its emittance and energy spread >CalcSigma.

The GUI has two panels for several input values affecting lifetime and for derived values >Output.

Coulomb lifetime is calculated from the effective acceptance, hower this includes only the physical, not the dynamic aperture limits >CalcAccEL. Bremsstrahlung lifetime uses the negative MA as used for Touschek lifetime too. Total lifetime is given as the inverse of the sum of the loss rates >CalcLifetime.

TrackLib.pas (46.2k)

---

uses OPAglobal, OpticPlot, OPAElements, MathLib;

At start, all linear elements are concatenated to matrices alternating with non-linear (or time dependent) kicks in order to speed up tracking >TrackinMatrix. Physical acceptances are calculated either from element apertures or from given apertures to estimate the maximum range for tracking >Acceptances. Both calculations have to be re-done when changing the reference momentum >Init_dpp. The available aperture in the (x, y)-plane is calculated for a set of rays to obtain the silhouette, i.e. projection of beam pipe apertures to the trackpoint as described in Appendix ?? >AmpKappa. Optics parameters at the Trackpoint are calculated, which may be located anywhere, even inside an element >TrackPoint.

Tracking proceeds turn by turn at fixed momentum >OneTurn (or with changing momentum based on a simple model of synchtron oscillation >OneTurn_S). Tracking one turn proceeds by repeated application of the matrix for a series of linear elements, followed by a non-linear (or time dependent) kick and a check for particle loss >TMatKick.

The procedures in the last third of the unit (from SineWindow to the end) are for signal processing, to calculate the FFT, interpolate frequencies and guess the related resonances.

11.9 Lattice Layout

OPAGeometry.pas/.lfm (63.2k)

---

uses OPAglobal, OpticPlot, MathLib, ASaux.

This unit displays the lattice layout, peforms geometric matching and exports various files. The orbit is calculated in 3D-space from element lengths, deflection and rotation angles >CalcOrbit. The orbit as curve in space is rotated and translated depending on the initial conditions >setDrawMode. The elements are made from faces, i.e. polygons in 3D-space >CalcFaces, which become polygons when projected to 2D-space of the image plane >CalcPoly. Changing the initial conditions does not change the faces themselves but applies the same translationa and rotation to all of them >TransPoly.

Up to now the elements are "flat", i.e. represented by a face of some length and width in the

midplane. If changing the initial angle, they look ugly. It would be straightforward to define boxes made from several faces, however this would also require some rendering algorithm for 3D-display.

Several files can be exported, among them a `.geo` file of polygons `>butListClick`, which can also be read `>ReadFiles` in order to show several lattice structures in one plot.

Geometric matching looks for lattice variables with names starting with "G", which control lengths and deflection or rotation angles of elements (in `>Start`) and runs an SVD minimizer to adjust the final (or initial) coordinates and angles of the lattice structure to a target value `>butMatchClick`.

Geometric matching works but is not well implemented yet. More checks are required, and also an "undo" button should be added.

11.10 Temporary

`OPACurrents.pas/.lfm` (10.3k)

---

uses `OPAglobal, ASaux.`

This unit calculates magnet currents using two tables for allocation (i.e. hardware type of the magnet in the lattice) and magnet calibration, which may be read with the lattice in `OPAGlobal> ReadAllocation, ReadCalibration`. Calculating the current from the strength parameters and v.v. is done in `OPAGlobal> getIfromK, getKfromI`. Due to non-linearity of the calibration curve, a bisection root finder is used [13].

The `OPACurrents` unit provides a GUI and writes a `.snap` file to be read by the storage ring control system. Among other data, this file also includes the matrices for tune change, if previously calculated by `OpticTune`, and for chromaticity change, calculated by `ChromLib> UpdateChromMatrix`.

The current calculation should be done here, not in `OPAGlobal`.

`opatest.pas` (37.3k)

---

uses `OPAglobal, OpticPlot, MathLib, ASaux.`

This unit is for temporarily needed procedures or for testing new stuff, which then, if it works well, is moved to another place.

In the present (June 21, 2025) version the unit contains, among others, three procedures for reading MAD `.mad`, MAD sequence `.seq` and ELEGANT `.lte` files. The procedures are yet incomplete but nevertheless facilitate reading these files.

References

[1] OPA Tutorial, https://andreas-streun.de/opa/tutorial2.pdf

[2] OPA userguide, https://andreas-streun.de/opa/opa4.pdf

[3] Inside OPA, https://andreas-streun.de/opa/inside.pdf

[4] Semantic versioning, https://semver.org/

[5] G. Slabaugh, Computing Euler angles from a rotation matrix, https://eecs.qmul.ac.uk/~gslabaugh/publications/euler.pdf.

[6] V. Ziemann and A. Streun, Equilibrium parameters in coupled storage ring lattices and practical applications, Phys. Rev. Accel. Beams, 25, 050703, 2022. https://link.aps.org/doi/10.1103/PhysRevAccelBeams.25.050703.

[7] K. Wille, Physik der Teilchenbeschleuniger und Synchrotronstrahlungsquellen, 1996.

[8] D. Edward and L.Teng, *Parametrization of linear coupled motion in periodic systems,* IEEE Trans.Nucl.Sci. 20, 885 (1973).

[9] D. Sagan, D. Rubin, *Linear Analysis of coupled lattices,* Physical Review Special Topics–Accelerators and Beams 2 (1999) 074001.

[10] J. Bengtsson, The sextupole scheme for the SLS, SLS-Note 9/97, http://slsbd.psi.ch/pub/slsnotes/sls0997.pdf

[11] Chun-xi Wang, Explicit formulas for 2nd-order driving terms due to sextupoles and chromatic effects of quadrupoles, ANL/APS/LS-330, 2012

[12] S. C. Leemann and A. Streun, Perspectives for future light source lattices incorporating yet uncommon magnets, Phys. ST Rev. Accel. Beams, 14, 030701, 2011.

[13] W. H. Press et al., Numerical Recipes in Pascal, Cambridge 1989.

[14] http://paulbourke.net/papers/conrec/

[15] http://rosettacode.org/wiki/Arithmetic_Evaluator/Pascal