Michigan State University

National Superconducting Cyclotron Laboratory

COSY INFINITY

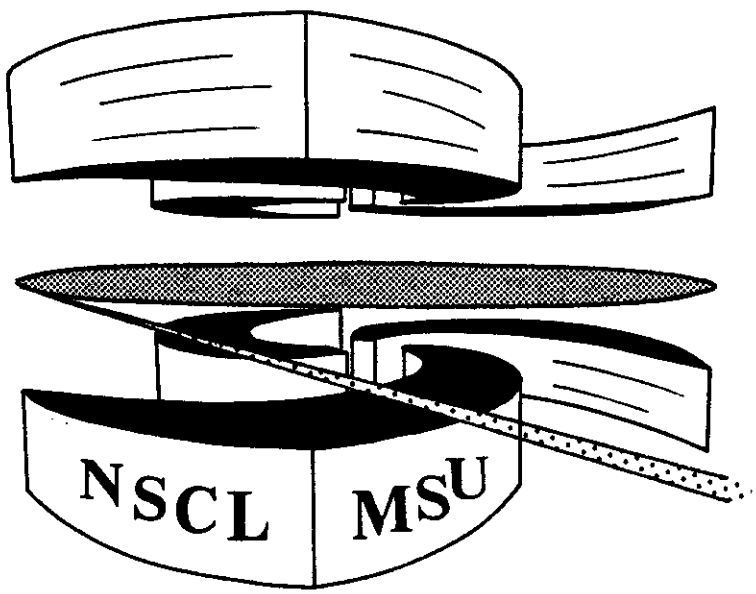
VERSION 4

USER'S GUIDE

AND

REFERENCE MANUAL

M. BERZ



MSUCL-771

MAY 1991

COSY INFINITY
Version 4

User's Guide
and
Reference Manual [1]

M. Berz

Department of Physics and Astronomy
and National Superconducting
Cyclotron Laboratory
Michigan State University
East Lansing, Mi 48824

[1] Supported in Part by the U.S. National Science Foundation, Grant No. PHY 89-13815

Abstract

This is a reference manual for the arbitrary order beam physics code COSY INFINITY. It is current as of July 5, 1991. COSY INFINITY is a powerful new generation code to study and design of beam physics systems including accelerators, spectrometers, beamlines, electron microscopes, and glass optical systems. At its core it is using differential algebraic (DA) methods, which allow a systematic calculation of arbitrary order effects of arbitrary particle optical elements. At the same time, it allows the computation of dependences on system parameters, which is often important and can also be used for fitting.

COSY INFINITY has a full structured object oriented language environment. This provides a simple interface for the casual user. At the same time, it offers the demanding user a very flexible and powerful tool for the study and design of systems. Elaborate optimizations are easily customized to the problem. The inclusion of new particle optical elements is straightforward.

COSY INFINITY provides a powerful environment for efficient utilization of DA techniques. The power and generality of the environment is perhaps best demonstrated by the fact that all the physics routines of COSY INFINITY are written in its own input language and can be printed on a handful of pages.

Altogether, the uniqueness of COSY lies in the ability to handle high order maps, and the ability to compute maps of arbitrary systems. Furthermore, its powerful work environment adopts to any conceivable problem. Its interactive, flexible graphics helps visualize even complicated connections between quantities.

Contents

1 Before Using the Code

1.1 User's Agreement

A licence for the code COSY INFINITY can be requested through Michigan State University. Licenses for educational institutions and government agencies are provided free of charge. There is a charge for licenses to profit-making agencies. Key parts of the license agreement are the following.

Users are requested not to make the code available to others, but ask them to obtain it from us. We want to maintain a list of users to be able to send out regular updates, which will also include features supplied by other users.

The FORTRAN portions and the high-level COSY language portions of the code should not be modified without our consent. This does not include the addition of new optimizers and new graphics drivers as discussed in sections 7.1 and 7.2; however, we would like to receive copies of new routines for possible inclusion in the master version of the code.

Though we do our best to keep the code bug free and hope that it is so now, we do not mind being convinced of the contrary and ask users to report any errors. Users are also encouraged to make suggestions for upgrades, or send us their tools written in the COSY language.

If the user thinks the code has been useful, we would like to see this acknowledged by referencing some of the papers related to the code, for example [1, 2]. Finally, we do neither guarantee correctness nor usefulness of this code, and we are not liable for any damage, material or emotional, that results from its use.

By using the code COSY INFINITY, users agree to be bound by the above conditions.

1.2 How to Obtain Help and to Give Feedback

While this manual is intended to describe the use of the code as completely as possible, there will most likely arise questions that this manual cannot answer. Furthermore, we encourage users to contact us with any suggestions, criticism, praise, or other feedback they may have. We also appreciate receiving COSY source code for utilities users have written and find helpful.

We prefer to communicate by electronic mail. We can be contacted as follows:

Prof. Martin Berz
Department of Physics and Astronomy
Michigan State University

East Lansing, MI 48824
USA
Phone: 517-353-0899
FAX: 517-353-5967
email: BERZ@MSUNSCL.BITNET

1.3 How to Install the Code

The code for COSY INFINITY consists of the following files:

- FOXY.FOP
- DAFOX.FOP
- FOXFIT.FOP
- FOXGRAF.FOP
- COSY.FOX

If obtained by electronic mail, each of these files will be split into several pieces. The pieces can be identified by the filename followed by the part number in the subject; for example, DAFOX.FOP003 identifies the third part of the file DAFOX. The files then have to be reassembled before compilation.

FOXY.FOP is the compiler and executer of the COSY language. DAFOX.FOP contains the routines to perform operations with objects, in particular the differential algebraic routines. FOXFIT.FOP contains the package of nonlinear optimizers. FOXGRAF.FOP contains the available graphics output drivers. These four files are written in FORTRAN and have to be compiled and linked. COSY.FOX contains all the physics of COSY INFINITY, and is written in COSY INFINITY's own input language. It has to be compiled by FOXY as part of the installation process.

All the FORTRAN parts of COSY INFINITY are written in standard ANSI FORTRAN 77. However, certain aspects of FORTRAN 77 are still system dependent; in particular, this concerns the file handling. All system dependent features of COSY INFINITY are coded for various machines, including VAX/VMS, SUN/UNIX, IBM, and CRAY (which is not fully complete at this time).

The type of machine can be changed by selectively adding and removing comment identifiers from certain lines. To go from VAX to SUN, for example, all lines that have the identifier *VAX somewhere in columns 72 through 80 have to be commented, and all lines that have the comment *SUN in columns 1 through 4 have to be un-commented. To automatize this process, there is a utility FORTRAN program called VERSION that performs all these changes automatically. Should there be additional problems, a short

message to us would be appreciated in order to facilitate life for future users on the same system.

1.3.1 VAX systems

The FORTRAN source is by default compatible with VAX/VMS systems. Compilation should be done without any options. In order to link and run the code, it may be necessary to increase certain working set parameters. The following parameters, generated with the VMS command SHOW WORK, are sufficient:

```
Working Set       /Limit= 1024   /Quota= 2500    /Extent= 8192
Adjustment enabled    Authorized Quota= 2500  Authorized Extent= 8192
```

The code has to be linked with the local GKS object code. It can be executed on workstation with UIS graphics, with Xwindows graphics, and on terminals supporting Tektronix.

1.3.2 SUN systems

On SUN systems, all lines that contain the string *SUN in columns 72 to 80 should be un-commented, and all the lines containing the string *VAX in columns 72 to 80 should be commented. This can be done using the small program VERSION, which has to be adjusted manually to perform the proper file handling.

Compilation should be performed with the compiler option "-Bstatic".

The code should be linked to the local GKS object code. GKS on HP systems usually requires the use of include files in the beginning of FOXGRAF.FOP as well as in all subroutines. These include statements are contained in the HP version, but they have to be moved from column 6 to column 1, and if necessary the address of the libraries has to be changed. On workstations, the graphics can be utilized under Xwindows and Tektronix.

1.3.3 HP systems

On HP systems, all lines that contain the string *HP in columns 72 to 80 should be un-commented, and all the lines containing the string *VAX in columns 72 to 80 should be commented. This can be done using the small program VERSION, which has to be adjusted manually to perform the proper file handling.

Compilation should be performed with the compiler option setting static memory handling.

The code should be linked to the local GKS object code. On workstations, the graphics can be utilized under Xwindows and Tektronix.

1.3.4 IBM Mainframes

On IBM mainframe systems, all lines that contain the string *IBM in columns 72 to 80 should be un-commented, and all the lines containing the string *VAX in columns 72 to 80 should be commented. This can be done using the small program VERSION, which has to be adjusted manually to perform the proper file handling.

On IBM mainframes it may be desirable to use COSY INFINITY in a menu-driven way like most other programs in that environment. This can be achieved by using the following command procedure kindly supplied by Ghislain Roy.

```
/* RUN FOXY */

ARG  F1 T1 M1
Say 'FOXY called...'
IF FI='' THEN Do
   Say '** No input file specified'
   Exit
End

'FILEDEF 6  TERMINAL (lrecl 133)'
FI FOX     DISK F1   FOX   M1 '(LRECL 80 RECFM F)'
FI LIS     DISK F1   LIS   M1 '(LRECL 80 RECFM F)'
FI COD     DISK F1   COD   M1 '(LRECL 80 RECFM F)'
FI 10      DISK F1   OUT   M1 '(LRECL 80 RECFM F)'
FI DEB     DISK F1   DEB   M1 '(LRECL 80 RECFM F)'

LOAD FOXY DAFOX FOXFIT '(NOMAP CLEAR )'
START

'FILEDEF  * CLEAR'
Exit rc
```

1.3.5 IBM PC

COSY INFINITY can be run on most IBM PC systems with 386 processors. For the large version allowing compuations of very high orders, the memory should be 16 or 32 megabytes to limit extensive disk paging. It should also be possible to install COSY on 286 systems. It is essential that the compiler can address arrays of lengths exceeding 64 k. An arithmetic coprocessor is highly recommended on 382 and 286 systems.

Because of the large number of different FORTRAN compilers available for IBM PC systems, it is not possible to maintain versions for all of them. There are some compilers which accept VAX FORTRAN, in which case the VAX version is a good starting point. The SUN version will also be a good starting point. These versions can be generated with the programm VERSION.

We do expect to have a complete version for the IBM PC based on the Lahey compiler available soon. This version was created by Phil Meads.

1.3.6 CRAY

On CRAY machines, all lines that contain the string *CRAY in columns 72 to 80 should be un-commented, and all the lines containing the string *VAX in columns 72 to 80 should be commented. This can be done using the small program VERSION, which has to be adjusted manually to perform the file handling properly.

Since the latest version of COSY INFINITY has not been explicitly tested on CRAYs yet, additional changes may be necessary.

1.3.7 Possible Memory Limitations

Being based on FORTRAN, which does not allow dynamic memory allocation, COSY INFINITY has its own memory management within a large FORTRAN COMMON block. On machines supporting virtual memory, the size of this block should not present any problem. On some other machines, it may be necessary to scale down the length. This can be achieved by changing the parameter LMEM at all occurrences in FOXY.FOP, DAFOX.FOP and FOXGRAF.FOP to a lower value. Values of around 200 000 should be enough for many applications, which brings total system memory down to about 4 Megabytes.

In the case of limited memory resources, it may also be necessary to scale down the lengths of certain variables in COSY.FOX to lower levels. In particular, this holds for the variables MAP and SCR which are defined at the very beginning of COSY.FOX. Possible values for the length are values down to about 500 for work through around fifth order. For higher orders, larger values are needed.

1.4 How to Avoid Reading This Manual

The input of COSY INFINITY is based on a programming language which is described in detail in section 6 beginning on page 43. The structure and features are quite intuitive, and we are confident that one can quickly pick up the key ideas following some examples.

COSY INFINITY is written in this language, and all particle optical elements and control features are invoked by calls to library procedures written in this language. A detailed description of these features is provided in sections 3 and 4.

Section 5 beginning on page 33 gives several examples for problems occurring in the computation and analysis of particle optical systems. Reading these sections should enable the user to get a head start in using COSY INFINITY. An excellent source of information is also the demonstration file DEMO.FOX.

For sophisticated problems or the development of customized features, the user may find it helpful to study section 7 beginning on page 50. The appendix beginning on page 58 contains a complete list of all data types and operations as well as all intrinsic functions and procedures available in the COSY language. Finally, the few pages of the listing of COSY INFINITY can be consulted for existing structures and programming ideas.

1.5 New Features of Version 4

This version of COSY INFINITY contains several new features which are outlined below. With very minor exceptions, it is downward compatible to the previous version, so any user deck for version 4 should run under version 5. The changes include

- The code now computes chromaticities and other parameter tune shifts directly without normal form, based on the new algorithm in [3], resulting in a very significant speed increase. There is now also a parameter dependent fixed point routine.
- The code now contains the new, purely DA based normal form algorithm [4] which leads to more efficient treatment. Damped Systems can also be studied.
- Due to changes in memory management, the code uses noticeably less memory
- Several features were optimized for speed. In particular, problems requiring only one or two phase space pairs benefit from a speed gain.
- The speed of several GKS-based graphics functions has been improved.
- The computation speed for round lenses and the high precision fringe field mode has been increased considerably. In some cases, a gain of three orders of magnitude has been reported for high order calculations.
- Several system dependent features like file handling have been adjusted for various systems, resulting in a speedier installation.
- A few minor bugs and inconveniences have been removed; for details, contact us.
- The manual now has an extensive index and contains more practical examples

- There are now utilities to open and close files by name and specify a status. Details can be found under the intrinsic procedures OPEN and CLOSE in the index. There is also a utility that returns CPU time called CPUSEC; details can be found in the index. This feature is system and is currently only available on the VAX.
- There are now various kinds of Wien Filters (E cross B devices) available in the code. For details, consult the respective sections.
- The LaTeX picture mode has been substantially improved; the limitations to a small number of fixed slopes in LaTeX has been circumvented. Now the accuracy and resolution of LaTeX pictures is as good as the accuracy of the other graphics modes.

1.6 Features Under Development

In the near future, the following new features are anticipated to be available in the code:

- Provide a tool to read beamlines in MAD standard
- Provide a tool to plot any quantity as it varies along the beamline
- Better fast fringe field approximations
- DA-based optimization, resulting in speed gains
- Spin and radiation dynamics
- Strict Nekoroshev-type estimates for long term stability

Please contact us if you have any other suggestions or wishes.

2 What is COSY INFINITY

The design and analysis of particle optical systems is quite intimately connected with the computer world. There are numerous more or less widespread codes for the simulation of particle optical systems. Generally, these codes fall into two categories. One category includes ray tracing codes which use numerical integrators to determine the trajectories of individual rays through external and possibly internal electromagnetic fields. The core of such a code is quite robust and easy to set up; for many applications, however, certain important information can not be directly extracted from the mere values ray coordinates. Furthermore, this type of code is often quite slow and does not allow extensive optimization.

The other category of codes are the mapping codes, which compute Taylor expansions to describe the action of the system on phase space. These codes are usually faster than integration codes, and the expansion coefficients often provide more insight into the system. On the other hand, the orders of the map, which are a measure of the accuracy of the approach, have been limited to third order [5, 6, 7] and fifth order [8, 9]. Furthermore, traditional mapping codes have only very limited libraries for quite standardized external fields and lack the flexibility of the numerical integration techniques. In particular, fringe fields can only be treated approximately.

2.1 Algorithms and Implementation

Recently we could show that it is indeed possible to have the best of both worlds: using the new differential algebraic techniques, any given numerical integration code can be modified such that it allows the computation of Taylor maps for arbitrarily complicated fields and to arbitrary order [10, 11, 12, 13]. An offspring of this approach is the computation of maps for large accelerators where often the system can be described by inexpensive, low order kick integrators [14, 15].

The speed of this approach is initially determined by the numerical integration process. Recently it has been possible to use DA techniques to overcome this problem: DA can be used to automatically emulate numerical integrators of very high orders in the time step, yet at the computational expense of only little more than a first order integrator [10, 11]. This technique is very versatile, works for a very large class of fields, and the speeds obtained are similar to classical mapping codes.

In order to make efficient use of DA operations in a computer environment, it has to be possible to invoke the DA operations from a language. In the conventional languages used for numerical applications it is not possible to introduce new data types and operations on them. Only recently have object oriented languages been developed which routinely have such features. One such language which is slowly gaining ground is C++; FORTH is another example which has been around for a longer time.

There are strong reasons to stay within the limits of a FORTRAN environment, however. Firstly, virtually all software in this field is written in this language, and the desire to interface to such software almost requires the use of FORTRAN. Furthermore, there are extensive libraries of support software which are only slowly becoming available in other languages, including routines for nonlinear optimization and various graphics packages. Finally, the necessity for portability is another strong argument for FORTRAN; virtually every machine that is used for numerical applications, starting from personal computers, continuing through the workstation and mainframe world to the supercomputers, has a FORTRAN compiler.

So it seemed natural to stay within this world, and this lead to the development of the DA precompiler [16]. This precompiler allows the use of a DA data type within otherwise standard FORTRAN by transforming arithmetic operations containing DA

variables into a sequence of calls to subroutines. This technique has been extensively used [11, 12, 17, 18, 19, 20]. It was particularly helpful that one could use old FORTRAN tracking codes and just replace the appropriate real variables by DA variables to very quickly obtain high order maps.

2.2 The User Interface

On the other end of the problems using an accelerator code is the command language of the code and the description of the beamlines. Various approaches have been used in the past, starting from coding numbers as in the old versions of TRANSPORT [5] over more easily readable command structures like in TRIO [6], GIOS [7, 21], COSY 5.0 [8, 18] and MARYLIE [22] to probably the most complete, standardized commands of MAD [23].

COSY INFINITY approaches this problem by offering the user a full programming language; in fact, the language is so powerful that all the physics of COSY INFINITY was written in it, and the results fit on a few pages.

The question is which language to select. For ease of use, it should have a simple syntax. For the user demanding special-purpose features, it should be powerful. It should allow direct and complex interfacing to FORTRAN routines, and it should allow the use of DA as a type. Finally, it should be widely portable. Unfortunately, there is no language readily available that fulfills all these requirements, so COSY INFINITY contains its own language system.

The problem of simplicity yet power has been quite elegantly solved by the PASCAL concept. In addition, this concept allows compilation in one pass and no linking is required. This facilitates the connection of the user input, which will turn out to be just the last procedure of the system, with the optics program itself.

To be machine independent, the output of the compilation is not machine code but rather an intermediate code that can be easily interpreted. For the same reason, it is essential to write the source code of the compiler in a very portable language. We chose FORTRAN for the compiler, even though clearly it is considerably easier to write it in a recursive language.

For reasons of speed it is helpful to allow the splitting of the program into pieces, one containing the optics program and one the user commands. While the PASCAL philosophy does not have provisions for linking, it allows the splitting of the input at any point. For this purpose, a complete momentary image of the compilation status is written to a file. When compilation continues with the second portion, this image is read from the file, and compilation continues in exactly the same way as without the splitting.

The full syntax of the COSY language is described in detail in section 6 beginning on

page 43. Most of the syntax will become apparent from the detailed examples supplied in the following sections, and we think that it is possible to write most COSY inputs without explicitly consulting the language reference.

3 Computing Systems with COSY

In this section we want to describe some core features of COSY's ion optics environment. This provides the backbone for practical use in particle optics. We assume that the reader has a fundamental knowledge about particle optics, and refer to the literature, for example [24, 25, 26, 27, 28, 29].

3.1 General Properties of the COSY Language Environment

The physics part of COSY INFINITY is written in its own input language. In this context, most commands are just calls to previously defined procedures. If desired, the user can create new commands simply by defining procedures of his own. All commands within COSY INFINITY consist of two letters which are abbreviations for two words describing the action of the procedure. This idea originated in the GIOS language [7, 21], and many commands of COSY INFINITY are similar to respective commands in GIOS.

Particle optical systems and beamlines are described by a sequence of calls to procedures representing individual elements. The supported particle optical elements can be found in section 3.3 beginning on page 20; section 5.7 beginning on page 41 shows how to generate new particle optical elements.

In a similar way, elements can be grouped, which is described in section 5.3 beginning on page 35. Besides the commands describing particle optical elements, there are commands to instruct the code what to do.

3.2 Control Commands

All user commands for COSY INFINITY are contained in a file which is compiled by FOXY. The first command of the file must be

INCLUDE COSY ;

which makes all the compiled code contained in COSY.FOX known to the user input. The user input itself is contained in the COSY procedure RUN. Following the syntax of the COSY language described in section 6, all commands thus have to be included between the statements

PROCEDURE RUN ;

and

ENDPROCEDURE ;

In order to execute the commands, the ENDPROCEDURE statement has to be followed by the call to the procedure,

RUN ;

and the command to complete the COSY input file,

END ;

Like any language, the COSY language supports the use of variables and expressions which often simplifies the description of the system. For the declaration of variables, see section 6.

The first command sets up the DA tools and has to be called before any DA operations, including the computation of maps, can be executed. The command has the form

OV <order> <phase space dimension> <number of parameters> ;

and the parameters are the maximum order that is to occur as well as the dimensionality of phase space (1,2 or 3) and the number of system parameters that are requested. If the phase space dimensionality is 1, then only the x-a motion is computed; if it is 2, y-b motion is computed as well, obviously at a slightly higher computation time. If it is 3, the time of flight and chromatic effects are computed also.

The number of parameters is the number of additional quantities besides the phase space variables that the final map shall depend on. This is used in connection with the "maps with knobs" discussed in section 5.2 on page 34 and to obtain mass and charge dependences if desired, and it is also possible to compute energy dependence without time-of-flight terms at a reduced computational expense.

The order is arbitrary and denotes the maximum order that computations can be performed in. It is possible to change the computation order at run time using the command

CO < order > ;

however, the new order can never exceed the one set in **OV**. Note that the computation time naturally increases drastically for higher orders. Under normal circumstances, orders should not exceed ten very much.

3.2.1 Defining the Beam

All particle optical coordinates are relative to a reference particle which can be defined with the command

RP < kinetic energy in MeV > < mass in amu > < charge in units > ;

For convenience, there are two procedures that set the reference particle to be protons or electrons:

RPP < kinetic energy in MeV >

RPE < kinetic energy in MeV >

For the masses of the proton and electron and all other quantities in COSY, the values in [30] have been used. Finally, there is a command that allows to set the reference particle from the magnetic rigidity in Tesla meters:

RPM < magnetic rigidity in Tm > < mass in amu > < charge in units > ;

The command

SB < PX > < PA > < PY > < PB > < PT > < PD > ;

sets half widths of the beam in the x, a, y, b, t and d directions of phase space, and the command

SP < P1 > < P2 > < P3 > < P4 > < P5 > < P6 > ;

sets the maxima of up to six parameters that can be used in connection with the maps with knobs 5.2 beginning on page 34.

3.2.2 The Computation of Maps

COSY INFINITY has a global variable called MAP that contains the accumulated transfer map of the system. Each particle optical element that is invoked updates the momentary contents of this global variable.

The following command is used to prepare the computation of maps. It sets the transfer map to unity. It can also be used again later to re-initialize the map.

UM ;

The command

SM < name > ;

saves the momentary transfer matrix to the array name, which has to be specified by the user. The array can be specified using the **VARIABLE** command of the COSY

language (see section 6). It could have the form

VARIABLE < name > 1000 8 ;

which declares a one dimensional array with eight entries. Each entry can hold a maximum of 1000 16 byte blocks, which should be enough for calculations of at least seventh order. The command

AM < name > ;

applies the previously saved map <name> to the momentary map. AM and PM are particularly helpful for the handling of maps of subsystems that are expensive to calculate. In particular in the context of optimization, often substantial amounts of time can be saved by computing certain maps only once and then re-use them during the optimization. The command

PM < unit > ;

prints the momentary transfer matrix to specified unit. Unit 6 corresponds to the screen. The command

RM < unit > ;

reads a map generated by COSY INFINITY from the specified unit and applies it to the momentary transfer map. Often a significant amount of computer time can be saved by computing certain submaps ahead of time and storing them either in a variable or a file. In particular this holds for maps which are expensive to compute, for example the ones of electrostatic cylindrical lenses.

We note that another important technique to save computation time in the computation and manipulation of maps is the use of maps with knobs discussed in section 5.2 beginning on page 34.

It is also possible to extract individual matrix elements of transfer maps. This is achieved with the COSY function

ME (<phase space variable >,< element identifier >)

The element identifier follows TRANSPORT notation; for example, ME(1,12) returns the momentary value of the matrix element (x,xa).

3.2.3 The Computation of Trajectories

Besides the computation of maps, COSY can also trace rays through the system. The trajectories of these rays can be plotted or their coordinates printed. If rays are selected, they are pushed through every new particle element that is invoked. Note that COSY can also push rays through maps repetitively and display phase space plots. This uses different methods and is discussed in section 4.5 beginning on page 32.

The following command sets a ray that is to be traced through the system. The parameters are the eight particle optical coordinates

SR <X> <A> <Y> <B> <T> <D> <G> <Z> ;

It is also possible to automatically set an ensemble of rays. This can be achieved with the command

ER <NX> <NA> <NY> <NB> <NT> <ND> <NG> <NZ> ;

Here NX, NA ... denote the number of rays in the respective phase space dimension. The ray coordinates are equally spaced according to the values set with the command **SB**, which has to be called before **SE**. In case any of the N's is 1, only rays with the respective variable equal to 0 will be shown. Note that the total number of rays is given by NX ·...· NZ, which can become rather large if many entries are different from 1.

The command

CR ;

clears all the rays previously set. The command

PR < unit > ;

prints the momentary coordinates of the rays to the specified unit. Unit 6 corresponds to the screen. Note that using the **WRITE** command of the COSY language, it is also possible to print any other quantity of interest either to the screen or to a file.

3.2.4 Plotting System and Trajectories

Besides computing matrices and rays, COSY also allows to plot the system or any part of it and the rays going through it. The command

BP ;

defines the beginning of a section of the system that is to be plotted, and the command

EP ;

defines the end of the section. The command

PP < unit > < phi > < theta > ;

plots the system to unit. Following the convention of printing graphics objects discussed in section 7.2 beginning on page 53, positive units produce a low-resolution ASCII plot of 80 columns by 24 lines, which does not require any graphics packages. Negative units correspond to various graphics standards.

The picture of the trajectories and elements is fully three dimensional and can be viewed from different angles. Phi=0 and Theta=0 correspond to the standard x projection; Phi=0 and Theta=90 correspond to the y projection; and Phi=90 and Theta=0 correspond to viewing the rays along the beam.

For use on workstations, there is also an abbreviated way to produce both an x projection and a y projection simultaneously. The command

PG < I > ;

produces both x and y pictures. I is the workstation mode, 1 denoting VMS, 2 denoting XWINDOWS; 0 produces low-resolution ASCII plots.

3.3 Supported Elements

In this section we present a list of all elements available in COSY. The elements range from standard multipoles and sectors over glass lenses and electromagnetic cylindrical lenses to a general element, which allows the computation of the map of any element from measured field data. The maps of all elements can be computed to arbitrary order and with arbitrarily many parameters.

Elements based on so-called strong focusing like multipoles and sectors can be computed with their fringing fields or without, which is the default. Section 3.3.4 beginning on page 23 explains how to turn on and off various fringe field computation modes.

The simplest particle optical element, the field- and material free drift, can be applied to the map with the command

DL < length > ;

The element

CB ;

changes the bending direction of bending magnets and deflectors. Initially, the bending direction is clockwise. The procedure CB changes it to counterclockwise, and each additional CB switches it to the other direction.

COSY supports a large ensemble of other particle optical elements, and it is very simple to add more elements. The following subsections contain a list of momentarily available elements.

3.3.1 Multipoles

COSY supports magnetic and electric multipoles in a variety of ways. There are the following magnetic multipoles:

MQ < length > < flux density at pole tip > < aperture > ;

MH < length > < flux density at pole tip > < aperture > ;

MO < length > < flux density at pole tip > < aperture > ;

MD < length > < flux density at pole tip > < aperture > ;

MZ < length > < flux density at pole tip > < aperture > ;

which let a magnetic quadrupole, sextupole, octupole, decapole or duodecapole act on the map. There is also a superimposed multipole for multipole strengths up to order five:

M5 < length > < BQ >< BH >< BO >< BD >< BZ >
< aperture > ;

And finally, there is a general superimposed magnetic multipole with arbitrary order multipoles:

MM < length > < MA > < NMA > < aperture > ;

Contrary to the previous procedure, the arguments now are the array MA and the number NMA of supplied multipole terms.

Similar procedures are available for electrostatic multipoles

EQ < length > < voltage at pole tip > < aperture > ;

EH < length > < voltage at pole tip > < aperture > ;

EO < length > < voltage at pole tip > < aperture > ;

ED < length > < voltage at pole tip > < aperture > ;

EZ < length > < voltage at pole tip > < aperture > ;

which let an electric quadrupole, sextupole, octupole, decapole or duodecapole act on the map. The strengths of the multipoles are described by their voltage in kV. There is an electric multipole

E5 < length >< EQ >< EH >< EO >< ED >< EZ >
< aperture > ;

which lets a superimposed electric multipole with components EQ through EZ act on the map, and there is the procedure

EM < length > < EA > < NEA > < aperture > ;

which lets a general electrostatic multipole with arbitrary order multipoles act on the map.

3.3.2 Bending Elements

Bending elements play a central role in particle optical systems. COSY INFINITY supports both magnetic and electrostatic elements including so called combined function elements with superimposed multipoles. In the case of magnetic elements, edge focusing and higher order edge effects are also supported.

The next commands let an inhomogeneous combined function bending magnet and a combined function electrostatic deflector act on the map:

MS < radius > < angle > < aperture > < n_1 >< n_2 >< n_3 >< n_4 >< n_5 > ;

ES < radius > < angle > < aperture > < n_1 >< n_2 >< n_3 >< n_4 >< n_5 > ;

The radius is in meters, the angle in degrees, and the aperture is in meters and corresponds to half of the gap width. The indices n_i describe the midplane radial field dependence which is given by

$$F(r) = F_0 \cdot [1 - \sum_{i=1}^{5} n_i \cdot (\frac{x}{r})^i]$$

where r is the bending radius. Note that an electric cylindrical condenser has $n_1 = 1$, $n_2 = -1$, $n_3 = 1$, $n_4 = -1$, $n_5 = 1$, etc, and an electric spherical condenser has $n_1 = 1$, $n_2 = -2$, $n_3 = 3$, $n_4 = -4$, $n_5 = 5$, etc. Homogeneous dipole magnets have $n_i = 0$. The element

DI < radius > < angle > < aperture > < ϵ_1 >< ϵ_2 > ;

lets a homogeneous dipole with entrance edge angle ϵ_1 and exit edge angle ϵ_2 act on the map. All angles are in degrees, the radius is in m, and the aperture is half of the gap width. Positive edge angles correspond to weaker x focusing. Finally, there is a very general combined function bending magnet with shaped entrance and exit edges

MC < radius > < angle > < aperture > < N > < S1 > < S2 > < n > ;

Here N is an array of n multipole strengths, and S1 and S2 are arrays containing the n coefficients s_1, ... s_n of two n-th order polynomials describing the shape of the entrance and exit edges as

$$S(x) = s_1 \cdot x + ... + s_n \cdot x^n$$

Again positive zeroth order terms entail weaker x focusing. Note that the computation of the edge effects is performed in the usual kick approximation and is active regardless of the fringing field mode. In the case of soft-edge fringing fields, the results will be approximations of the real fringe field effects.

Especially for small deviations from perpendicular entry, the accuracy of this technique is very high. A fully accurate treatment of the fringing field effects of a combined function magnet with nonlinear edge effects can be performed using the general particle optical element 3.3.6.

3.3.3 Wien Filters

Besides the purely magnetic and electric bending elements, there are routines for superimposed electric and magnetic deflectors, so-called Wien Filters or E cross B devices. The simplest Wien Filter consists of homogeneous electric and magnetic fields which are superimposed such that the reference trajectory is straight. This element is called by

WF < radius$_1$ > < radius$_2$ > < length > < aperture >

The radii describe the bending power of the magnetic and electric fields, respectively. The strengths are chosen such that each one of them alone would deflect the beam with the specified radius. For positive radii, the electric field bends in the direction of positive x, and the magnetic field bends in the direction of negative x. For equal radii, there is no net deflection. There is also a combined function Wien Filter:

WC < radius$_1$ > < radius$_2$ > < length > < aperture > < NE > < NM > < n > ;

Here NE and NM describe the inhomogeneity of the electric and magnetic fields, respectively via

$$\cdot F(x) = F_0 \cdot [1 + \sum_{i=1}^{n} N(i) \cdot x^i]$$

3.3.4 Fringe Fields

COSY INFINITY allows the computation of fringe field effects of particle optical elements. In the default, fringe field effects are not taken into account. However, it is possible to compute all effects of particle optical elements including fringe fields with varying degrees of accuracy and varying computational expense. The command

FR < mode > ;

sets the fringing field calculation mode. If mode is 0, no fringe fields are used. Mode 1 entails approximate fringe fields with an accuracy comparable to the fringe field integral method. Mode 2 entails fringe fields with a higher accuracy, but also a higher computation time. Mode 3 entails completely accurate fringe fields at an even higher computation time.

It is also possible to calculate fringing fields of elements alone. If mode is set to -1, only entrance fringe fields are computed, and if mode is set to -2, only exit fringe fields of all listed elements are computed. In both cases, the computation is very accurate.

If fringing fields are very important, this often allows the computation of a particular fringing field once and for all which can then be stored and re-used using the commands **SM** and **AM** or also **PM** and **RM** (see section 3.2.2 beginning on page 17).

3.3.5 Cylindrical Electromagnetic Lenses

COSY INFINITY also allows the use of a variety of cylindrical lenses, in which focusing effects occur only due to fringe field effects. The simplest such element consists of only one ring of radius d that carries a current I. The field of such a ring is given by

$$B(s) = \frac{\mu_0 I}{2d} \cdot \frac{1}{\left(1+(z/d)^2\right)^{3/2}} \quad .$$

This current ring is represented by the procedure

CMR < I > < d > ;

A magnetic field of more practical significance is that of the so-called Glaser lens, which represents a good approximation of the fields generated by strong magnetic lenses with short magnetic pole pieces [29]. The lens is characterized by the field

$$B(s) = \frac{B}{1+(s/d)^2} \quad ,$$

where B is the maximum field in Tesla and d is the half-width of the field. The Glaser lens is invoked by calling the procedure

CML < B > < d > ;

A third magnetic round lens available in COSY is the solenoid with the following field distribution:

$$B(s) = B \cdot \tanh[s/d] - \tanh[(s-l)/d]$$

where B is the field strength inside the solenoid, d is its aperture and l its length. The solenoid is invoked by the procedure

CMS < B > < d > < l > ;

There is a third magnetic round lens with a Gaussian potential

$$A(s) = A_0 \cdot \exp[-(s/d)^2]$$

which is invoked with the procedure

CMG < A > < d > ;

Besides the magnetic round lenses, there are various electrostatic round lenses. The element

CEL < L > < V > < d > < c > ;

lets an electrostatic lens consisting of three tubes act on the map. The geometry of the lens consists of three coaxial tubes with identical radii d, of which the outer ones are on ground potential and the inner one is at potential V. The length of the middle tube is L, and the distance between the central tube and each of the outside tubes c. Such an arrangement of three tubes can be shown to produce an axis potential of the form

$$V(s) = \frac{V}{2\omega/dc}\left(\ln\frac{\cosh(\omega(s+L/2)/d)}{\cosh(\omega(s+L/2+c)/d)} + \ln\frac{\cosh(\omega(s-L/2)/d)}{\cosh(\omega(s-L/2-c)/d)}\right) ,$$

where the value of the constant ω is 1.315. For details, refer to [27]. An often used approximation for the electrostatic lenses is described by a potential distribution of the following form

$$V(s) = V_0 \cdot \exp[-(s/d)^2] \ .$$

A lens with this field can be invoked by calling the routine

CEG < V_0 > < d >

3.3.6 General Particle Optical Element

In this section, we present a procedure that allows the computation of an arbitrary order map for a completely general optical element whose fields are described by measurements along the independent variable s. Its use ranges from special measured fringe fields over dedicated electrostatic lenses to the computation of maps for cyclotron orbits. It can also be used to custom build new elements that are frequently used (see section 5.7 on page 41).

GE < n > < m > < S > < H > < V > < W >

lets an arbitrary particle optical element act on the map. The element is characterized by arrays specifying the values of multipole strengths at the n positions along the independent variable contained in the array S. The array H contains the corresponding curvatures at the positions in S. V and W contain the electric and magnetic scalar potentials in S.

The elements in V and W have to be DA variables containing the momentary derivatives in the x direction (variable 1) and s direction (variable 2). m is the order of the s-derivatives.

3.3.7 Glass Lenses and Mirrors

COSY INFINITY also allows the computation of higher order effects of general glass optical systems. At the present time, it contains elements for spherical lenses and mirrors, parabolic lenses and mirrors, and general surface lenses and mirrors, where the surface is described by a polynomial. There is also a prism. All these elements can be combined to systems like particle optical elements, including misalignments. The dispersion of the glass can be treated very elegantly by making the index of refraction a parameter using the function **PARA**. The routines were written by Meng Zhao.

The command

GLS < R1 > < R2 > < N > < d > ;

lets a spherical glass lens act on the map. R1 and R2 are the radii of the spheres; positive radii means that the center of the sphere is to the right. N is the index of refraction, and d is the thickness. The command

GL < P1 > < P2 > < N > < d > ;

lets a glass lens whose surface is specified by two polynomials act on the map. P1 and P2 are two dimensional arrays containing the coefficients of the polynomials in x and y that describe the s position of the entrance and exit surface as a function of x and y. N is the index of refraction, and d is the thickness. The command

GP < PHI1 > < PHI2 > < N > < d > ;

lets a glass prism act on the map. PHI1 and PHI2 are the entrance and exit angles, N is the index of refraction and d is the thickness.

Besides the refractive glass optical elements, there are mirrors. The command

GMS < R > ;

lets a spherical mirror with radius R act on the map. The command

GMP < R > ;

lets a parabolic mirror with central radius of curvature R act on the map. The command

GMF ;

lets a flat mirror act on the map. The command

GM < P > ;

lets a general glass mirror act on the map. P is a two dimensional array containing the coefficients of the polynomial in x and y that describes the surface.

3.4 Misalignments

The differential algebraic concept allows a particularly simple and systematic treatment of misalignment errors in optical systems. Such an error is represented by a coordinate change similar to the one discussed in section 4.3. COSY offers three different misalignment commands. The command

SA < DX > < DY > ;

The first commands offsets the optic axis by DX in x direction and DY in y direction. DX and DY are counted positive if the optic axis is shifted in direction of positive x and y, respectively. The command

TA < AX > < AY > ;

represents a tilt of the optic axis by an angle of AX in x direction and AY in y direction. AX and AY are counted positive if the direction of tilt is in the direction of positive x and y, respectively. The command

RA < ANGLE > ;

represents a rotation of the optic axis around ANGLE measured in degrees. ANGLE is counted positive if the rotation is counterclockwise if viewed in the direction of the beam.

In order to simulate a single particle optical element that is offset in positive x direction, it is necessary to have the element preceeded by an axis shift with negative value and followed by an axis shift with positive value. Similarly simple geometric considerations tell how to treat single tilted and rotated elements.

The misalignment routines can also be used to study beams that are injected off the optical axis of the system. In this case, just one of each misalignment commands is necessary at the beginning of the system.

We note that the misalignment routines, like most other COSY routines, can be called both with real number and differential algebraic arguments, in particular using the **PARA** argument (see section 5.2). The first case allows the simulation of a fixed given misalignment, whereas the second case allows to compute the map depending on the misalignment.

In the first case, the values of the computed transfer matrix are only approximate if **SA** and **TA** are used. The accuracy increases with decreasing misalignments and

increasing calculation orders. For the study of misalignments of elements, the actual accuracy is usually rather high since the values of the misalignments are usually very small. In the case of a deliberate offset of the beam, for example for the study of injection and extraction processes, it may be necessary to increase the computation order to obtain accurate results. In the second case, the results are always accurate. The command **RA** always produces accurate results in both cases.

4 Analyzing Systems with COSY

4.1 Image Aberrations

Very often not the matrix elements of the transfer map are of primary significance, but rather the maximum size of the resulting aberration for the phase space defined with **SB** and the parameters defined with **SP**. COSY provides two tools to obtain the aberrations directly. The command

PA < unit >

prints all aberrations to unit in a similar way as **PM**. If not all aberrations are of interest, the COSY function

MA (<phase space variable >,< element identifier >)

returns the momentary value of the aberration. For example, **MA**(1,12) returns the momentary value of the aberration due to the matrix element (x,xa).

4.2 Twiss Parameters, Tunes and Chromaticities

Instead of by their transfer matrices, the linear motion in particle optical systems is often described by the tune and twiss parameters. These quantities being particularly important for repetitive systems, they allow a direct answer to questions of linear stability, beam envelopes, etc. In many practical problems, their dependence on parameters is very important. For example, the dependence of the tune on energy, the chromaticity, is a very crucial quantity for the design of systems. Using the maps with knobs, they can be computed totally automatically without any extra effort. The command

TP < MU > ;

computes the tunes. MU is an array defined by the user. If the system is run with parameters, the tunes will automatically depend on them. This allows a very direct computation of chromaticities. Note that COSY INFINITY can also compute amplitude dependent tune shifts in the framework of normal form theory. This is described in detail in section 4.6 beginning on page 33.

Often the Twiss parameters as well as the fixed points are also of importance. They can be computed using the command

GT < MM > < F > < NU > < ALPHA > < BETA > < GAMMA > < R > ;

It computes the fixed point F, the tunes NU and the Twiss parameters ALPHA, BETA and GAMMA from the map MM. It also computes the determinant of the linear map R, which is relevant for damped systems. Note again that as soon as the computation of the map is performed with parameters, all the quantities computed by GT automatically contain the dependence on these knobs. So a direct computation of the parameter dependent fixed point or the energy dependence of ALPHA is possible.

Finally there is also the routine

ST < I > < NU > < ALPHA > < BETA > < GAMMA > ;

which allows to set the linear part of the matrix to the specified values. This is often helpful to set a certain pre-specified orientation of the phase space ellipse.

4.3 Coordinates and their Changes

COSY INFINITY performs all its calculations in the following scaled coordinates:

$$\begin{array}{llll} r_1 = x, & & r_2 = a = p_x/p_0, & \\ r_3 = y, & & r_4 = b = p_y/p_0, & \\ r_5 = l = -(t-t_0)\gamma/(1+\gamma) & & r_6 = \delta_K = (K-K_0)/K_0) & \\ r_7 = \delta_m = (m-m_0)/m_0 & & r_8 = \delta_z = (z-z_0)/z_0 & \end{array}$$

The first six variables form three canonically conjugate pairs in which the map is symplectic. p_0, K_0 and γ are the momentum, kinetic energy, and total energy over c^2, respectively. m and z denote mass and charge, respectively.

For comparison and other reasons, it is often helpful to express the map in other coordinates, for example the ones used in TRANSPORT [5] and GIOS [7, 21]. The routine

PT < unit > ;

prints the map in Transport and GIOS coordinates to unit. We want to point out that in the differential algebraic concept, it is particularly simple to perform such nonlinear coordinate changes to arbitrary orders. All that is required is to evaluate the algebraic formulas describing the transformation of coordinates in differential algebra. In the case of the transformation from COSY a and b to TRANSPORT coordinates x' and y', these relationships read

$$
\begin{aligned}
x' &= a \cdot \left(F - (a^2 + b^2)\right)^{-1/2} \\
y' &= b \cdot \left(F - (a^2 + b^2)\right)^{-1/2} \\
a &= x' \cdot \left(1 - (x'^2 + y'^2)/F\right)^{-1/2} \\
b &= y' \cdot \left(1 - (x'^2 + y'^2)/F\right)^{-1/2}
\end{aligned}
$$

where

$$F = (1 + \delta_m)^2 \cdot \frac{\eta(2 + \eta)}{\eta_0(2 + \eta_0)}$$

$$\eta = \frac{K_0}{m_0 c^2} \cdot \frac{1 + \delta_k}{1 + \delta_m}$$

and K_0, m_0 are kinetic energy and mass of the reference particle and δ_k and δ_m are the energy and mass deviations of the particle under consideration.

In order to print maps in yet different coordinates, the user can make a procedure that begins with a unity map, applies the transformation to COSY coordinates, applies the COSY map, and then applies the transformation back to the original coordinates.

4.4 Symplectic Representations

In this section, we will present two different representations for symplectic maps, each one of which has certain advantages. Particle optical systems described by Hamiltonian motion satisfy the symplectic condition

$$M \cdot J \cdot M^t = J, \text{ or alternatively } M \cdot J = (M \cdot J)^t$$

where M is the Jacobian Matrix of partial derivatives of $\mathcal{M}$, and J has the form

$$J = \begin{pmatrix} 0 & 0 & 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{pmatrix}$$

As long as there is no damping, all particle optical systems are Hamiltonian, up to possibly computation errors if they are generated numerically. There is a COSY function that checks if a certain map satisfies the symplectic condition:

SE < M > ;

Here < M > is an array of DA quantities describing the map. Note Note that the momentary value of the transfer map is stored in the global COSY variable MAP.

Symplectic maps can be represented by at least one of four so-called generating functions in mixed variables:

$$F_1(q_i, q_f) \text{ satisfying } (\vec{p}_i, \vec{p}_f) = J \cdot \vec{\nabla} F_1$$

$$F_2(q_i, p_f) \text{ satisfying } (\vec{p}_i, \vec{q}_f) = J \cdot \vec{\nabla} F_2$$

$$F_3(p_i, q_f) \text{ satisfying } (\vec{q}_i, \vec{p}_f) = J \cdot \vec{\nabla} F_3$$

$$F_4(p_i, p_f) \text{ satisfying } (\vec{q}_i, \vec{q}_f) = J \cdot \vec{\nabla} F_4$$

In the generating function representation there are no interrelationships between the coefficients due to symplecticity like in the transfer map, so the generating function representation is more compact. Furthermore, it is often an important tool for the symplectification of tracking data. The command

GF < F > < I > < IER > ;

attempts to compute the I th generating function of the momentary system map. If IER is equal to zero, this generating function exists and is contained in F. If IER is nonzero, it does not exist.

Another redundance free representation of symplectic transfer maps is the Dragt-Finn factorization [31, 32, 10]. . It is based on Lie operators of the form

$$\exp(: f_i :) = 1 + : f_i : + \frac{: f_i :}{2} + \ldots$$

where the colon denotes a poisson bracket waiting to happen, i.e. $: f_i : g = \{f_i, g\}$. The map describing the system is given by the action of the operators on the vector $(q_1, p_1, q_2, p_2, \ldots, q_n, p_n)$. The Dragt-Finn factorization then has the form

$$\mathcal{M}(\vec{x}) =_n (L \ \exp(: f_3 :) \exp(: f_4 :) \ldots \exp(: f_{n+1} :)) \vec{x}$$

where each of the f_i is a homogenous polynomial in the phase space variables of exact order i, and L is a linear matrix.

Besides this factorization, there are various others that are similar and have certain advantages [10]. They are

$$\mathcal{M}(\vec{x}) =_{2^{n+1}} (L \ \exp(: f_{3,3} :) \exp(: f_{4,5} :) \exp(: f_{6,9} :) ... \exp(: f_{(2^n+2),(2^{n+1}+1)}) \vec{x}.$$

$$\mathcal{M}(\vec{x}) =_{2^{n+1}} (\exp(: f_{2^n+2,2^{n+1}+1}) ... \exp(: f_{6,9} :) \exp(: f_{4,5} :) \exp(: f_{3,3} :) L) \vec{x}$$

and

$$\mathcal{M}(\vec{x}) =_n (\exp(: f_{n+1} :) ... \exp(: f_5 :) \exp(: f_4 :) \exp(: f_3 :) L) \vec{x}$$

As shown in [10], it is one of the strong points of the map representation and the differential algebraic techniques that the computation of these Dragt-Finn factorizations is possible to arbitrary order with a relatively simple algorithm. It is actually much easier to compute them from the map than using Lie algebraic techniques alone. The command

LF < C > < M > < F > < NF > < I > ;

computes the factorization from the momentary transfer map. C contains the constant part, M the linear part and the array F with NF entries contains the f_i from above. I is the identifier of the factorization following the above numbering.

4.5 Repetitive Tracking

COSY allows very efficient repetitive tracking of particles through maps. The command

TR < N > < Np > < d > < I > < M > < IU >;

tracks the momentary particles through the momentary map for the required number of iterations N. After each Np iterations, the coordinates of the phase space projection I are output to unit IU. If coordinates get larger than d, the particle is considered lost and ignored in the future. For M=1, the tracking is performed without symplectification algorithms; for Mode=2, the tracking is performed with symplectification using the global polynomial-type generating function. (This feature is not available yet).

The algorithm used for tracking is highly optimized for speed. Using the vector data type for particle coordinates, it works most efficiently if many particles are tracked simultaneously. On scalar machines, optimum efficiency is obtained when more than about 20 particles are tracked simultaneously. On vector machines, the algorithm vectorizes completely, and for best efficiency, the number of particles should be a multiple of the length of the hardware vector.

In both cases, logistics overhead necessary for the bookkeeping is almost completely negligible, and the computation time is almost entirely spent on arithmetic. It is also worth mentioning that using an optimal tree transversal algorithm, zero terms occurring in a map do not contribute to computation time.

4.6 Amplitude Tune Shifts and Normal Form

COSY INFINITY contains an implementation of the DA normal form algorithm described in [4]. This replaces the COSY implementation of the somewhat less efficient and less general mixed DA-Lie normal form [17]. Normal Form algorithms provide nonlinear transformations to new coordinates in which the motion is simpler. They allow the determination of pseudo invariants of the system, and they are the only tool so far to compute amplitude tune shifts. As pointed out in [3], chromaticities and parameter dependent tune shifts can be computed more directly. Their computation is described in section 4.2 beginning on page 28. The command

NF < EPS > < MA > ;

computes the normal form transformation map MA of the momentary transfer map. Since the normal form algorithm has a small denominator problem, it is not always possible to perform a transformation to coordinates in which the motion is given by circles. The variable EPS sets the minimum size of a resonance denominator that is not removed. The command

TS < MU > ;

employs the normal form algorithm to compute all the tune shifts of the system, both the ones depending on amplitude and the ones depending on parameters like chromaticities, which alone can be computed more efficiently as shown in section 4.2. MU is a one dimensional array that on return will contain the tunes as they depend on parameters and amplitude. Note that in some cases when the system is on or very near a resonance or is even unstable, the normal form algorithm fails because of a small denominator problem. In this case, the respective tunes will be returned as zero. This also happens sometimes if the map is supposed to be symplectic yet is slightly off because of computational inaccuracies.

5 Examples

5.1 A Simple Sequence of Elements

After having discussed the particle optical elements and features available in COSY INFINITY in the previous sections, we now discuss the computation of maps of simple systems.

We begin with the computation of the transfer map of a quadrupole doublet to tenth order. Here the COSY input resembles the input of many other optics codes [8, 7].

```
INCLUDE COSY ;
PROCEDURE RUN ;
```

```
   OV 10 2 0 ;      {order 10, phase space dim 2, # of parameters 0}
   RP 10 4 2 ;      {kinetic energy 10 MeV, mass 4 amu, charge 2}
   UM ;             {sets map to unity}
   DL .1 ;          {drift of length .1 m}
   MQ .2 .1 .05 ;   {quad; length .2 m, field .1 T, aperture .05 m}
   DL .1 ;
   MQ .2 -.1 .05 ;  {defocussing quad}
   DL .1 ;
   PM 11 ;          {prints map to unit 11}
   ENDPROCEDURE ;
RUN ; END ;
```

5.2 Maps with Knobs

The DA approach easily allows to compute maps not only depending on phase space variables, but also on system parameters. This can be very helpful for different reasons. For example, it directly tells how sensitive the system is to errors in a particular quantity. In the same way it can be used to find out ideal positions to place correcting elements. Furthermore, it can be very helpful for the optimization of systems, and sometimes very fast convergence can be achieved with it (for details, see section 7.1).

In the context of COSY INFINITY, the treatment of such system parameters or knobs is particularly elegant.

In the following example, we compute the map of a system depending on the strength of one quadrupole. The COSY function PARA(I) is used, which identifies the quantity as parameter number I by turning it into an appropriate DA vector.

```
INCLUDE COSY ;
PROCEDURE RUN ;
   OV 5 2 1 ;                {order 5, phase space dim 2, parameters 1}
   RP 10 4 2 ;               {sets kinetic energy, mass and charge}
   UM ;
   DL .1 ;
   MQ .2 .1*PARA(1) .05 ;    {quadrupole; now field is a DA quantity}
   DL .1 ;
   MQ .2 -.1 .05 ;
   DL .1 ;
   PM 11 ;                   {prints map depending on quad strength}
   ENDPROCEDURE ;
RUN ; END ;
```

In this context it is important that the COSY language supports freedom of types at

compile time; so the second argument of the quad can be either real or DA. For details, consult section 6.

The idea of maps with knobs can also be used to compute the dependence on the particle mass and charge as well as on energy in case time of flight terms are not needed. In the following example, the map of the quad doublet is computed including the dependence on energy, mass and charge.

```
INCLUDE COSY ;
PROCEDURE RUN ;
   OV 5 2 3 ;                {order 5, phase space dim 2, parameters 3}
   RP 10*PARA(1) 4*PARA(2) 2*PARA(3) ;    {sets kinetic energy, mass
                                           and charge as DA quantities}
   UM ;
   DL .1 ;
   MQ .2 .1 .05 ;
   DL .1 ;
   MQ .2 -.1 .05 ;
   DL .1 ;
   PM 11 ;                   {prints map with dependence on energy,
                              mass and charge, to unit 11}
   ENDPROCEDURE ;
RUN ; END ;
```

Note that since the phase space variables occupy the first six independent DA variables, parameters are always identified by numbers beginning with 7.

5.3 Grouping of Elements

Usually it is necessary to group a set of elements together into a cell. For example, since most circular accelerators are built of several at least almost identical cells, it is desirable to refer to the cell as a block. Similar situations often occur for spectrometers or microscopes if similar quad multiplets are used repetitively.

Grouping is easily accomplished in COSY by just putting the elements into a procedure. In the following example, the strength of a quadrupole in the cell of an accelerator is adjusted manually such that the motion in both planes is stable. Since the motions are stable if the two traces are less than two in magnitude, the map is printed to the screen which allows a direct check.

```
INCLUDE COSY ;
PROCEDURE RUN ; VARIABLE QS 1 ;
  PROCEDURE CELL Q H1 H2 ;          {defines a cell of a ring}
```

```
    DL .3 ; DI 10 20 .1 0 0 ; DL .1 ; MH .1 H1 .05 ;
    DL .1 ; MQ .1 Q .05 ;     DL .3 ; MH .1 H2 .05 ;
  ENDPROCEDURE ;
  OV 1 2 0  ; RPP 1000 ;     {third order, one GeV protons}
  QS := .1 ;                   {set initial value for quad}
  WHILE QS#0 ; WRITE 6 ' GIVE QS ' ; READ 5 QS ;
    UM ; CELL QS 0 0 ;  PM 6 ; WRITE 6 ME(3,3) ;
    ENDWHILE ;
ENDPROCEDURE ; RUN ; END ;
ENDPROCEDURE ; RUN ; END ;
```

Obviously, such groupings can be nested if necessary, and parameters on which the elements in the group depend can be passed freely. Note that calling a group entails that all elements in it are executed; so grouping is not a means to reduce execution time, but a way to organize complicated systems into easily manageable parts. Reduction of execution time can be achieved by saving maps of subsystems that do not change using SM and AM discussed above.

5.4 Optimization

One of the most important tasks in the design of optical systems is the optimization of certain parameters of the system to meet certain specifications. Because of the importance of optimization, there is direct support from the COSY language via the **FIT** and **ENDFIT** commands. COSY provides several FORTRAN based optimizers; a detailed description of the optimizers available in COSY can be found in section 7.1.

In the first example we illustrate a simple optimization task: to fit the strengths of the quadrupoles of a symmetric triplet to perform stigmatic point-to-point imaging. To monitor the optimization process, the momentary values of the quad strengths and the objective function are printed to the screen. Furthermore, a graphic display of the system at each step of the optimizer is displayed in two graphic windows, one for each phase space projection, creating a movie-like effect. At the end, the final picture of the x projection of the system is printed in LaTeX picture format for inclusion in this manual.

```
INCLUDE COSY ;
PROCEDURE RUN ;
  VARIABLE Q1 1 ; VARIABLE Q2 1 ; VARIABLE OBJ 1 ;
  PROCEDURE TRIPLET A B ;
    MQ .1 A .05 ; DL .05 ; MQ .1 -B .05 ; DL .05 ; MQ .1 A .05 ;
    ENDPROCEDURE ;
  OV 1 2 0 ;
  RP 1 1 1 ;
  SB .15 .15 .15 .15 0 0 ;
```

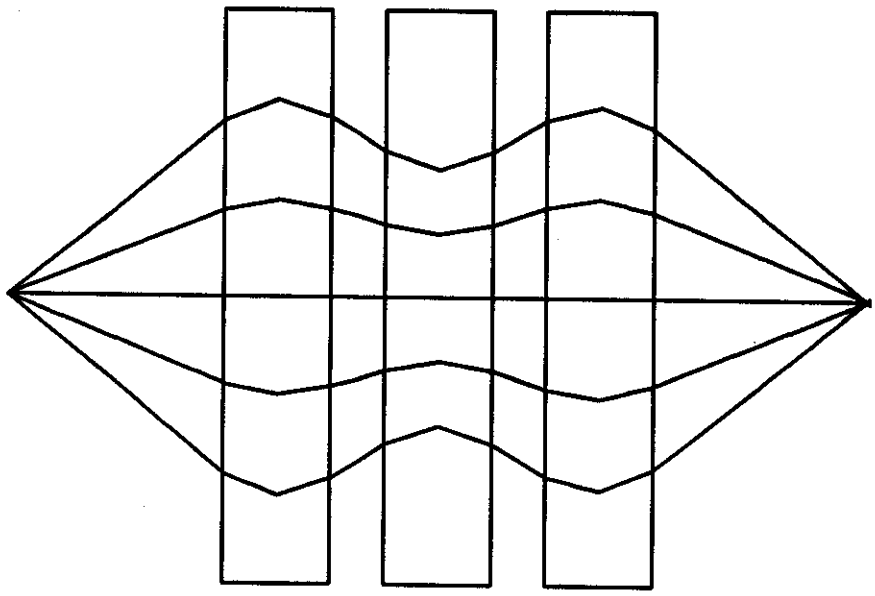


Figure 1: COSY LaTeX picture of the stigmatically focusing system

```
Q1 := .5 ; Q2 := .5 ;
FIT Q1 Q2 ;
   UM ; CR ; ER 1 3 1 3  1 1 1 1 ;
   BP ; DL .2 ; TRIPLET Q1 Q2 ; DL .2 ; EP ;
   PP -1 0 0 ; PP -51 0 90 ;
   OBJ := ABS(ME(1,2))+ABS(ME(3,4)) ;
   WRITE 6 'STRENGTHS Q1, Q2, OBJECTIVE FUNCTION: ' Q1 Q2 OBJ ;
   ENDFIT 1E-5 1000 1 OBJ ; PP -7 0 0 ; PP -7 0 90 ;
ENDPROCEDURE ; RUN ; END ;
```

The picture of the system after optimization is shown in figure 1. Note that the LaTeX file of this picture produced by COSY has been copied into the LaTeX source of this manual and is now a permanent part of it.

Besides providing "canned" optimization strategies, the COSY language allows to follow one's own path of optimizing a system, which typically consists of several runs with varying parameters and subsequent optimizations.

In the following example, the goal is to vary several parameters of the system manually, fit the quad strengths, and then look at the spherical aberrations. This process is repeated by inputting different values for the parameters until the spherical aberrations have been reduced to a satisfactory level. When this is achieved, the picture of the

system is printed in LaTeX format, which is identified with unit -7.

```
INCLUDE COSY ;
PROCEDURE RUN ;
   VARIABLE Q1 1 ; VARIABLE Q2 1 ; VARIABLE L1 1 ; VARIABLE L2 1 ;
   VARIABLE OBJ 1 ; VARIABLE ISTOP 1 ;
   PROCEDURE TRIPLET ;
      UM ; CR ; ER 1 4 1 4   1 1 1 1 ; BP ;
      DL L1 ; MQ .1 Q1 .05 ; DL L2 ; MQ .1 -Q2 .05 ;
      DL L2 ; MQ .1 Q1 .05 ; DL L1   ; EP ; PP -1 0 0 ;
      ENDPROCEDURE ;
   OV 3 2 0 ; RP 1 1 1 ; SB .08 .08 .08 .08 0 0  ; ISTOP := 1 ;
   WHILE ISTOP#0 ;
      WRITE 6 ' GIVE VALUES FOR L1, L2: ' ; READ 5 L1 ; READ 5 L2 ;
      Q1 := .5 ; Q2 := .5 ; CO 1 ;
      FIT Q1 Q2 ; TRIPLET ; OBJ := ABS(ME(1,2))+ABS(ME(3,4)) ;
         ENDFIT 1E-5 1000 1 OBJ ;
         CO 3 ; TRIPLET ;
         WRITE 6 ' SPHERICAL ABERRATION FOR THIS SYSTEM: ' ME(1,222) ;
         WRITE 6 ' CONTINUE SEARCH? (1/0) ' ; READ 5 ISTOP ;
      ENDWHILE ; PP -7 0 0 ; PP -7 0 90 ;
   ENDPROCEDURE ; RUN ; END ;
```

This example shows how it is possible to phrase more complicated interactive optimization tasks in the COSY language. One can even go far beyond the level of sophistication displayed here; by nesting sufficiently many WHILE, IF and LOOP statements, it is often possible to optimize a whole system in one interactive session without ever leaving COSY. For example, the first order design in [33] which is subject to quite a number of constraints and requires a sophisticated combination of trial and optimization was performed in this way.

5.5 Normal Form, Tune Shifts and Twiss Parameters

The following example shows the use of normal form methods and parameter dependent Twiss parameters for the analysis of a repetitive system. For the sake of simplicity, we choose here a simple FODO cell that is described by the procedure CELL. The map of the cell is computed to fifth order, with the energy as a parameter. In the cell itself, the quadrupole strength is another parameter.

As a first step, the parameter dependent tunes are computed and written to unit 7, following the algorithm in [3]. Next follow the tunes depending on parameters and amplitude; this is done with DA normal form theory [4]. Finally, several other quantities

and their parameter dependence are computed using the procedure TP. They include the parameter dependent fixed point, the parameter dependent Twiss parameters, as well as the parameter dependent damping (which here is unity because no radiation effects are taken into account).

```
INCLUDE COSY ;
   PROCEDURE RUN  ;
      VARIABLE A 100 3 ; VARIABLE B 100 3 ; VARIABLE G 100 3 ;
      VARIABLE R 100 3 ; VARIABLE MU 100 3 ; VARIABLE F  100 10 ;
      VARIABLE MN 2000 8 ; VARIABLE MA 2000 8 ;
      PROCEDURE CELL ;
         DL .1 ; DI 1 45 .1 0 0 ; DL .1 ; MQ .1 -.1*PARA(2) .1 ; DL .2 ;
         ENDPROCEDURE ;
      OV 5 2 2 ; RP 1*PARA(1) 1 1 ; UM ; CELL ;
      TP MU ; WRITE 7 ' DELTA DEPENDENT TUNES '          MU(1) MU(2) ;
      TS MU ; WRITE 7 ' DELTA AND EPS DEPENDENT TUNES ' MU(1) MU(2) ;
      GT MAP F MU A B G R  ;
      WRITE 7 ' DELTA DEPENDENT FIXED POINT ' F(1) F(2) F(3) F(4) ;
      WRITE 7 ' DELTA DEPENDENT ALPHAS '       A(1) A(2) ;
      WRITE 7 ' DELTA DEPENDENT BETAS '        B(1) B(2) ;
      WRITE 7 ' DELTA DEPENDENT GAMMAS '       G(1) G(2) ;
      WRITE 7 ' DELTA DEPENDENT DAMPINGS '     R(1) R(2) ;
      ENDPROCEDURE ; RUN ; END ;
```

5.6 Repetitive Tracking

In the following example, we want to study the nonlinear behaviour of a ring by a qualitative analysis of tracking data. The ring consists of 18 identical cells. Nine of these cells are packed into a half cell by the procedure HALFCELL. At execution, the system asks for the values of the strengths of the two hexapoles which influence its degree of nonlinearity. The tracking data for each setting are displayed and then also output in LaTeX format for inclusion in this manual. In order to keep the size of the LaTeX source file limited, only 100 turns were tracked for five particles.

```
INCLUDE COSY ;
PROCEDURE RUN ; VARIABLE QS 1 ; VARIABLE H1 1 ; VARIABLE H2 1 ; VARIABLE N 1 ;
   PROCEDURE CELL Q H1 H2 ;          {defines a cell of a ring}
      DL .3 ; DI 10 20 .1 0 0 ; DL .1 ; MH .1 H1 .05 ;
      DL .1 ; MQ .1 Q .05 ;     DL .3 ; MH .1 H2 .05 ;
   ENDPROCEDURE ;
   PROCEDURE HALFRING Q H1 H2 ; VARIABLE I 1 ;
      LOOP I 1 9 ; CELL Q H1 H2 ; ENDLOOP ; ENDPROCEDURE ;
```

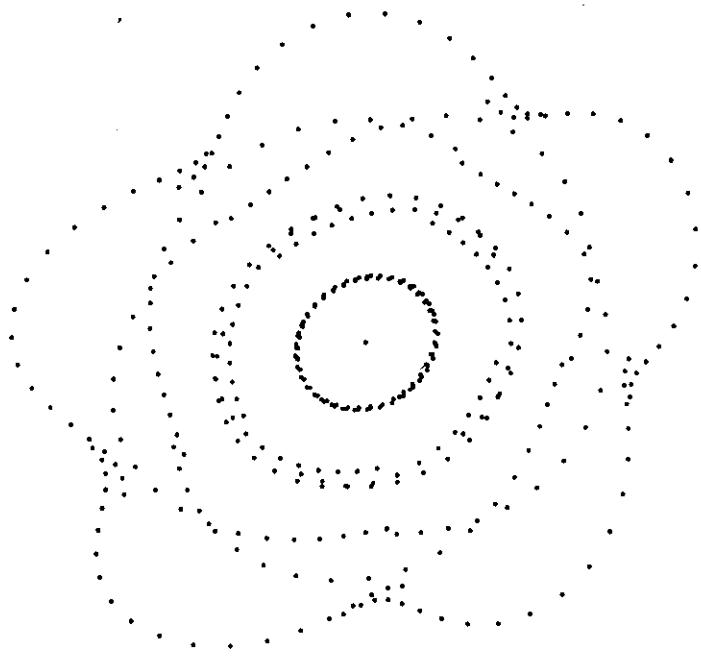


Figure 2: COSY LaTeX tracking picture

```
OV 3 2 0  ; RPP 1000 ;      {third order, one GeV protons}
QS := -.05 ; H1 := .01 ;
WHILE H1#0 ; WRITE 6 ' GIVE HEXAPOLE STRENGTHS ' ; READ 5 H1 ; READ 5 H2 ;
   UM ; HALFRING QS H1 H2 ;
   WRITE 6 ' GIVE NUMBER OF TURNS ' ; READ 5 N ;
   SR .005 0 .005 0 0 0 0 0 ;
   SR .01  0 .01  0 0 0 0 0 ;
   SR .015 0 .015 0 0 0 0 0 ;
   SR .02  0 .02  0 0 0 0 0 ;
   TR N 1  1 2  .03 .002   1 -1  ; CR ;
   SR .005 0 .005 0 0 0 0 0 ;
   SR .01  0 .01  0 0 0 0 0 ;
   SR .015 0 .015 0 0 0 0 0 ;
   SR .02  0 .02  0 0 0 0 0 ;
   TR N 1  1 2  .03 .002   1 -7 ;  ENDWHILE ;
ENDPROCEDURE ; RUN ; END ;
```

5.7 Introducing New Elements

When looking into the physics part of COSY INFINITY, it becomes apparent that all particle optical elements described above are nothing but procedures written in the COSY language. Due to the openness of the approach, users can construct their own particle optical elements.

Here we want to show how a user can define his own particle optical element and work with it. As a first example, we begin with a skew quadrupole that is rotated against the regular orientation by the angle ϕ. The action of such a quad can be obtained by first rotating the map by $-\phi$, then let the quad act, and finally rotate back. All these steps are performed on the DA variable containing the momentary value of the transfer map, which is the global COSY array MAP. For the conversion of degrees to radians, the global COSY variable DEGRAD is used. Note that many important global variables of COSY are described in section 5.8.

```
INCLUDE COSY ;
PROCEDURE RUN ;
   PROCEDURE SQ PHI L B D ;    {computes the action of a skew quad}
      PROCEDURE ROTATE PHI ;         {local procedure for rotation}
      VARIABLE M 4 1000 ; VARIABLE I 1 ;
         M(1) :=  COS(PHI*DEGRAD)*MAP(1) + SIN(PHI*DEGRAD)*MAP(3) ;
         M(3) := -SIN(PHI*DEGRAD)*MAP(1) + COS(PHI*DEGRAD)*MAP(3) ;
         M(2) :=  COS(PHI*DEGRAD)*MAP(2) + SIN(PHI*DEGRAD)*MAP(4) ;
         M(4) := -SIN(PHI*DEGRAD)*MAP(2) + COS(PHI*DEGRAD)*MAP(4) ;
         LOOP I 1 4 ; MAP(I) := M(I) ; ENDLOOP ; ENDPROCEDURE ;
      ROTATE -PHI ; MQ L B D ; ROTATE PHI ; ENDPROCEDURE ;
   OV 5 2 0 ;
   UM ;
   DL .1 ;
   SQ -30 .2 .1 .1 ;
   DL .1 ;
   SQ 30 .2 .1 .1 ;
   PM 6 ;
   ENDPROCEDURE ; RUN ; END ;
```

It is clear that a similar technique can be used to study misaligned elements. In a similar way, it is easily possible to generate a "kick-environment" in COSY INFINITY, where every particle optical element is just represented by a kick in its center.

This technique is also useful in many other ways. For example, if a certain element is rather time consuming to compute, which can be the case with cylindrical lenses to high orders, one can write a procedure that computes the map of the element, including the dependence on some of its parameters, and saves the map somewhere. When called

again with different values, the procedure decides if the values are close enough to the old ones to just utilize the previously computed map with the parameters plugged in, or if it is necessary to compute the element again. In case the parameters are varied only slightly, a very significant speed up can be achieved in this way, yet for the user the procedure looks like any other element.

5.8 Introducing New Features

The whole concept of COSY INFINITY is very open in that it easily allows extensions for specific tasks. The user is free to provide his own procedures for particle optical elements or for many other purposes. To interface with COSY INFINITY most efficiently, it is important to know the names of certain key global variables, functions and procedures. Furthermore it is important to know that all quantities in COSY INFINITY are in SI units, with the exception of voltages, which are in kV.

For some applications, it is helpful to access some of COSY INFINITY's global variables. Since the physics of the code is written in its own language, all these variables are directly visible to the user. The first set of relevant global variables are the natural constants describing the physics. These variables are set after the routine RP is called and can be utilized for calculations by the user. The data are taken from [30]. In order to match other codes, the variables can be changed by the user in COSY.FOX if necessary.

AMU	Atomic Mass Unit	$1.6605402 \cdot 10^{-27}$ kg
AMUMEV	Atomic Mass Unit in MeV	931.49432 MeV
EZERO	The charge unit	$1.60217733 \cdot 10^{-19}$ C
CLIGHT	The speed of light	$2.99792458 \cdot 10^{8}$ m/s
PI	the value of π	computed as 4 atan (1)

The second set of variables describes the reference particle. These variables are updated every time the procedure **RP** is called.

E0	Energy in MeV
M0	Mass in AMU
Z0	Charge in units
V0	Velocity
P0	Momentum
CHIM	Magnetic Rigidity
CHIE	Electric Rigidity
ETA	Kinetic Energy over mc^2

Finally, there are the variables that are updated by particle optical elements:

MAP	Array of 8 DA vectors containing Map
RAY	Array of 8 VE vectors containing Coordinates
SPOS	Momentary value of the independent variable

COSY INFINITY contains several procedures that are not used explicitly by the user but are used internally for certain operations. Firstly, there are the three DA functions

DER(< n >,< a>)

INTEG(< n >,< a>)

PB (< a >,< b>)

which compute the DA derivation with respect to variable n, the integral with respect to variable n, and the Poisson bracket between a and b. Another helpful function is

NMON (< NO >,< NV >)

which returns the maximum number of coefficients in a DA vector in NV variables to order NO. An important procedure is

POLVAL < L > < P > < NP > < A > < NA > < R > < NR > ;

which lets the polynomial described by the NP DA vectors stored in the array P act on the NA arguments A, and the result is stored in the NR Vectors R. Note that the type of A is free; it can be either DA or CD, in which case the procedure acts as a concatenator, it can be real or complex, in which case it acts like a polynomial evaluator, or it can be of vector type VE, in which case it acts as a very efficient vectorizing map evaluator and is used for repetitive tracking.

6 The COSY Language

6.1 General Aspects

In this section we will discuss the syntax of the COSY language. A brief summary of the commands can be found in section 6.6 on page 49. It will become apparent that the language has the flavor of PASCAL, which has a particularly simple syntax yet is relatively easy to analyze by a compiler and rather powerful.

The language of COSY differs from PASCAL in its object oriented features. New data types and operations on them can easily be implemented by putting them into a language description file described in the appendix. Furthermore, all type checking is done at run time, not at compile time. This has significant advantages for the practical use of DA and will be discussed below.

Throughout this section, curly brackets like "{" and "}" denote elements that can be repeated.

Most commands of the COSY language consist of a keyword, followed by expressions and names of variables, and terminated by a semicolon. The individual entries and the semicolon are separated by blanks. The exceptions are the assignment statement, which does not have a keyword but is identified by the assignment identifier :=, and the call to a procedure, in which case the procedure name is used instead of the keyword.

Line breaks are not significant; commands can extend over several lines, and several commands can be in one line. To facilitate readability of the code, it is possible to include comments. Everything contained within a pair of curly brackets "{" and "}" is ignored.

Each keyword and each name consist of up to 32 characters, of which the first has to be a letter and the subsequent ones can be letters, numbers or the underline sign "_". The case of the letters is not significant.

6.2 Program Segments and Structuring

The language consists of a tree-structured arrangement of nested program segments. There are three types of program segments. The first is the main program, of which there has to be exactly one and which has to begin at the top of the input file and ends at the end. It is denoted by the keywords

BEGIN ;

and

END ;

The other two types of program segments are procedures and functions. Their beginning and ending are denoted by the commands

PROCEDURE <name> { <name > } ;

and

ENDPROCEDURE ;

as well as

FUNCTION <name> { <name > } ;

ENDFUNCTION ;

The first name identifies the procedure and function for the purpose of calling it. The optional names define the local names of variables that are passed into the routine. Like in other languages, the name of the function can be used in arithmetic expressions, whereas the call to a procedure is a separate statement.

Inside each program segment, there are three sections. The first section contains the declaration of local variables, the second section contains the local procedures and functions, and the third section contains the executable code. A variable is declared with the following command:

VARIABLE <name> <expression> { <expression> } ;

Here the name denotes the identifier of the variable to be declared. As mentioned above, the types of variables are free at declaration time. The next expression contains the amount of memory that has to be allocated when the variable is used. The amount of memory has to be sufficient to hold the various types that the variable can assume. To simplify the determination of the required memory, there are various functions that return the required lengths for certain types.

If the variable is to be used with indices as an array, the next expressions have to specify the different dimensions. Note the elements of an array can have different types. Thus it is possible to emulate most of the record concept found in PASCAL using arrays.

Note that different from PASCAL practice, names of variables that are being passed into a function or procedure do not have to be declared.

All variables are visible inside the program segment in which they are declared as well as in all other program segments inside it. In case a variable has the same name as one that is visible from a higher level routine, its name and dimension override the name and properties of the higher level variable of the same name for the remainder of the procedure and all local procedures.

The next section of the program segment contains the declaration of local procedures and functions. Any such program segment is visible in the segment in which it was declared and in all program segments inside the segment in which it was declared, as long as the reference is physically located below the declaration of the local procedure. Recursive calls are permitted. Altogether, the local and global visibility of variables and procedures follows standard structured programming practice.

The third and final section of the program segment contains executable statements. Among the permissible executable statements is the assignment statement, which has the form

<variable or array element> := <expression> ;

The assignment statement does not require a keyword. It is characterized by the assignment identifier :=. The expression is a combination of variables and array elements visible in the routine, combined with operands and grouped by parentheses, following common practice. Note that due to the object oriented features, various operands can be loaded for various data types, and default hierarchies for the operands can be given. Parentheses are allowed to override default hierarchies. The indices of array elements can themselves be expressions.

Another executable statement is the call to a procedure. This statement does not require a keyword either. It has the form

<procedure name> { <expression> } ;

The name is the identifier of the procedure to be called which has to be visible at the current position. The rest are the arguments passed into the procedure. The number of arguments has to match the number of arguments in the declaration of the procedure.

6.3 Flow Control Statements

Besides the assignment statement and the procedure statement, there are statements that control the program flow. These statements consist of matching pairs denoting the beginning and ending of a control structure and sometimes of a third statement that can occur between such beginning and ending statements. Control statements can be nested as long as the beginning and ending of the lower level control structure is completely contained inside the same section of the higher level control structure.

The first such control structure begins with

IF <expression> ;

which later has to be matched by the command

ENDIF ;

If desired, there can be an arbitrary number of statements of the form

ELSEIF <expression> ;

between the matching **IF** and **ENDIF** statements.

If there is a structure involving **IF**, **ELSEIF** and **ENDIF**, the first expression in the **IF** or **ELSEIF** is evaluated. If it is not of logical type, an error message will be issued. If the value is true, execution will continue after the current line and until the next **ELSEIF**, at which point execution continues after the **ENDIF**.

If the value is false, the same procedure is followed with the logical expression in the next **ELSEIF**, until all of them have been reached, at which point execution continues after the **ENDIF**. So at most one of the sections of code separated by **IF** and the matching optional **ELSEIF** and the **ENDIF** statements is executed.

The next such control structure consists of the pair

WHILE <expression> ;

and

ENDWHILE ;

If the expression is not of type logical, an error message will be issued. Otherwise, if it has the value true, execution is continued after the **WHILE** statement; otherwise, it is continued after the **ENDWHILE** statement. In the former case, execution continues until the **ENDWHILE** statement is reached. After this, it continues at the matching **WHILE**, where again the expression is checked. Thus, the block is run through over and over again as long as the expression has the proper value.

Another such control structure is the familiar loop, consisting of the pair

LOOP <name> <expression> <expression> {<expression >} ;

and

ENDLOOP ;

Here the first entry is the name of a visible variable which will act as the loop variable, the first and second expressions are the first and second bounds of the loop variable. If a third expression is present, this is the step size; otherwise, the step size is set to 1. Initially the loop variable is set to the first bound.

If the step size is positive or zero and the loop variable is not greater than the second bound, or the step size is negative and the loop variable is not smaller than the second bound, execution is continued at the next statement, otherwise after the matching **ENDLOOP** statement. When the matching **ENDLOOP** statement is reached after execution of the statements inside the loop, the step size is added to the loop variable. Then, the value of the loop variable is compared to the second bound in the same way as above, and execution is continued after the **LOOP** or the **ENDLOOP** statement, depending on the outcome of the comparison. Note that it is allowed to alter the value of the loop variable inside the loop, which allows a premature truncation of the loop.

The final control structure in the syntax of the COSY language allows nonlinear optimization as part of the syntax of the language. This is an unusual feature not found in other languages, and it could also be expressed in other ways using procedure calls. But the great importance of nonlinear optimization in applications of the language and the clarity in the code that can be achieved with it seemed to justify such a step. The structure consists of the pair

FIT <name> {<name>} ;

and

ENDFIT <expression> <expression> <expression> {<expression>} ;

Here the names denote the visible variables that are being adjusted. The first expression is the tolerance to which the minimum is requested. The second expression is the maximum number of evaluations of the objective function permitted. The third expression gives the number of the optimizing algorithm that is being used. For the various optimizing algorithms, see section 7.1. The following expressions are of real or integer

type and denote the objective quantities, the quantities that have to be minimized.

This structure is run through over and over again, where for each pass the optimization algorithm changes the values of the variables listed in the **FIT** statement and attempts to minimize the objective quantity. This continues until the algorithm does not succeed in decreasing the objective quantity anymore by more than the tolerance or the allowed number of iterations has been exhausted. After the optimization terminates, the variables contain the values corresponding to the lowest value of the objective quantity encountered by the algorithm.

6.4 Input and Output

Besides the commands just presented, there are commands for i/o. They appear as commands and not as procedure calls because they have variable number of arguments. They have the form

READ <expression> <name> ;

and

WRITE <expression> {<expression>} ;

Here the first expression stands for a unit number; unit 6 is the screen. Unit numbers can be associated with particular file names using the bf OPEN and **CLOSE** procedures. For details, consult the index. In the READ command, the name denotes the variable to be read. Note that right so far only real numbers can be read. In the WRITE command, the expressions following the unit are the output quantities.

Besides these commands, it is possible to save code in compiled form. This allows later inclusion in another program without recompiling. The command

SAVE <name> ;

saves the compiled code on the file name.bin. The command

INCLUDE <name> ;

includes the previously compiled code in name.bin. Each code may contain only one INCLUDE statement, and it has to be located at the very top of the file. The SAVE and INCLUDE statements allow to break the code into a chain of easily manageable pieces and increase compilation times considerably.

6.5 Error Messages

COSY distinguishes between five different kinds of error messages which have different meanings and require different actions to correct the underlying problem. The five types

of error messages are identified by the symbols **###**, **$$$**, **!!!**, **@@@** and ***. In addition, there are informational messages, denoted by ---. The meaning of the error messages is as follows:

###: This message denotes errors in the syntax of the user input. Usually a short message describing the problem is given, including the command in error. If this is not enough information to remedy the problem, the file <inputfile>.lis can be consulted. It contains an element-by-element listing of the user input, including the error messages at the appropriate positions.

$$$: This error message denotes runtime errors in a syntactically correct user input. Circumstances under which it is issued include array bound violations, type violations, missing initialization of variables, exhaustion of the memory of a variable, and illegal operations like division by zero.

!!!: This message denotes exhaustion of certain internal arrays in the compiler. Since the basis of COSY is FORTRAN which is not recursive and requires a fixed memory allocation, all arrays used in the compiler have to be previously declared. This entails that in certain cases of big programs etc., the upper limits of the arrays can be reached. In such a case the user is told which parameter has to be increased. The problem can be remedied by replacing the value of the parameter by a larger value and re-compiling. Note that all occurrences of the parameter have to be changed; usually the parameters occur under the same name in many subroutines.

@@@: This message describes a catastrophic error, and should never occur with any kind of user input, erroneous or not. It means that COSY has found an internal error in its code by using certain self checks. In the rare case that such an error message is encountered, the user is kindly asked to contact us and submit the user program.

***: This error message denotes errors in the use of COSY INFINITY library procedures. It includes messages about improper sequences and improper values for parameters.

In case execution cannot be continued usefully, a system error exit is produced. Depending on the system COSY is run on, this may include information about the status at the time of error. In order to be system independent, this is done by attempting to execute the computation of the root of a negative number.

6.6 List of Keywords

This section contains a complete list of keywords of the COSY language.

BEGIN		END
VARIABLE		
PROCEDURE		ENDPROCEDURE
FUNCTION		ENDFUNCTION
IF	ELSEIF	ENDIF
WHILE		ENDWHILE
LOOP		ENDLOOP
FIT		ENDFIT
WRITE	READ	
SAVE	INCLUDE	

7 Further Information

7.1 Optimization

Many problems in the design of particle optical systems require the use of nonlinear optimization algorithms. COSY INFINITY supports the use of nonlinear optimizers at its language level using the commands **FIT** and **ENDFIT** (see section 6 beginning on page 43). The optimizers for this purpose are given as FORTRAN subroutines. For a list of momentarily available optimizers, see section 7.1.1. Because of a relatively simple interface, it is also possible to include new optimizers relatively easily. Details can be found in section 7.1.2.

Besides the FORTRAN algorithms for nonlinear optimization, the COSY language allows the user to design his own optimization strategies depending on the problem. Some thoughts about problem dependent optimizers can be found in section 7.1.3.

7.1.1 Optimizers

The **FIT** and **ENDFIT** commands of COSY allow the use of various different optimizers supplied in FORTRAN to minimize an objective function that depends on several variables. For details on the syntax of the commands, we refer to section 6 beginning on page 43. The choice of the proper objective function is up to the user, and it sometimes influences the success or failure of the optimization attempt.

While many problems are directly minimizations of a single intuitive quantity, others often require the simultaneous satisfaction of a set of conditions. In this later case, it is clearly possible to construct a total objective function that has a minimum if and only if the conditions are satisfied; this can for example be achieved by writing the conditions in the form $f_i(\vec{x}) = 0$ and then forming the objective function as a sum of absolute values or a sum of squares. By using factors in front of the summands, it is possible to give more or less emphasis on certain conditions.

At the present time, COSY supports three different optimizers with different features and strengths and weaknesses to minimize such objective functions. In the following we present a list of the various optimizers with a short description of their strengths and weaknesses. Each number is followed by the optimizer it identifies.

1. The Simplex Algorithm
   This optimizer is suitable for rather general objective functions that do not have to satisfy any smoothness criteria. It is quite rugged and finds local (and sometimes global) minima in a rather large class of cases. In simple cases, it requires more execution time than algorithms for almost linear functions. Because of its generality it is often the algorithm of choice.

2. The Parabolic Minimizer
   This optimizer is particularly suitable for functions that are nearly quadratic in the variables. For suitable objective functions, it is extremely fast and reliable. It often also works for other functions, but with a lower success rate than the Simplex Algorithm and a degradation of speed. Note that the user has significant freedom in choosing the objective function, and it is often possible to formulate the problem such that it can be represented by a nearly quadratic function.

3. The Simulated Annealing Algorithm
   This algorithm is often able to find the total minimum for very complicated objective functions, especially in cases where all other optimizers fail. This comes at the expense of a very high and often prohibitive number of function evaluations. Often this algorithm is also helpful for finding start values for the subsequent use of other algorithms.

7.1.2 Adding an Optimizer

COSY INFINITY has a relatively simple interface that allows the addition of other FORTRAN optimizers. All optimizers that can be used in COSY must use "reverse communication". This means that the optimizer does not control the program flow, but rather acts as an oracle which is called repeatedly. Each time it returns a point and either requests that the objective function be evaluated at this new point, after which the optimizer is to be called again. This continues until the optimum is found, at which time a control variable is set to a certain value.

All optimizers are interfaced to COSY INFINITY via the routine FIT at the beginning of the file FOXFIT, which is the routine that is called from the code executer in FOXY. The arguments for the routine are as follows:

IFIT	→	identification number of optimizer
XV	↔	current array of variables
NV	→	number of variables
EPS	→	desired accuracy of function value
ITER	→	maximum allowed iteration number
IEND	←	status identifier

The last argument, the status identifier, communicates the status of the optimization process to the executer of COSY. As long as it is nonzero, the optimizer requests evaluation of the objective function at the returned point X. If it is zero, the optimum has been found up to the abilities of the optimizer, and X contains the point where the minimum occurs.

The subroutine FIT branches to the various supported optimizers according to the value IFIT. It also supplies the various parameters required by the local optimizers. To include a new optimizer merely requires to put another statement label into the computed GOTO statement and calling the routine with the proper parameters.

We note that when writing an optimizer for reverse communication, it is very important to have the optimizer remember the variables describing the optimization status from one call to the next. This can be achieved using the FORTRAN statement SAVE. If the optimizer can return at several different positions, it is also important to retain the information from where the return occurred.

In case a user interfaces an optimizer of his own into COSY, we would appreciate receiving a copy of the amended file FOXFIT in order to be able to distribute the optimizer to other users as well.

7.1.3 Problem Dependent Optimization Strategies

In the case of very complicated optimization tasks, the use of any optimization algorithm alone is often too restrictive and inefficient. In practice it is more often than not necessary to combine certain "hand-tuning" tasks with the use of optimizers or to just check the terrain by evaluating the objective function on a coarse multidimensional grid.

We want to point out that because of its language structure, COSY INFINITY gives the demanding user very far reaching freedom to taylor his own optimization strategy. This can be achieved by properly nested structures involving loops, while blocks or if blocks. Manual tuning can be performed by reading certain variables from the screen in a while loop and then printing other quantities of interest or even the system graphics to the screen.

Besides being powerful, these strategies have also proved quite economical. In most cases it is possible to reduce the number of required runs considerably by choosing appropriate combinations of interactive tuning and hard wired as well as self made optimization strategies. With some advance thought this at least in principle makes it possible to design even rather complicated systems with only one COSY run.

7.2 Graphics

The object oriented language on which COSY INFINITY is based supports graphics via the graphics object. This is used for all the graphics generated by COSY and allows a rather elegant generation and manipulation of pictures.

The operand "&" allows the merging of graphics objects, and COSY INFINITY has functions that return individual moves and draws and various other elementary operations which can be glued together with "&". For details, we refer to the appendix beginning on page 58.

7.2.1 Supported Graphics Drivers

COSY INFINITY allows to print graphics objects using a variety of drivers which are addressed by different unit numbers. When a graphics object is printed using COSY's WRITE or the procedure **PP**, positive unit numbers produce a low resolution 80 column by 24 lines ASCII output of the picture written to unit. Unit 6 again corresponds to the screen. This does not require any graphics devices and is often helpful for interactive design. Furthermore, it is by far the fastest way to print a picture. Besides the low resolution graphics, there are various high resolution output modes some of which require linking to certain graphics packages.

- positive: Low-Resolution ASCII output to respective unit; 6: screen.
- -1: GKS-based output to primary VMS/UIS workstation window
- -2: GKS-based Tektronix output
- -3: GKS-based X-Windows workstation output
- -4: GKS-based postscript output to POSTSCRIPT.PLT
- -5: GKS-based HP 7475 plotter output to file HP7475.PLT
- -6: GKS-based HP Paintjet / DEC JL250 to file HPPAINT.PLT
- -7: LaTeX picture mode output to file PICTURE.TEX
- -51 ... -60: GKS-based secondary VMS/UIS workstation windows output

- -61 ... -70: GKS-based secondary XWINDOWS workstation windows output
- -71: GKS-based secondary Tektronix output

Note that the LaTeX picture mode does not require linking to any graphics driver and is an option in any environment supporting LaTeX. The file PICTURE.TEX can either be processed directly with LaTeX and sent to a printer or it can be incorporated into a LaTeX document. For the development of new graphics output drivers, we refer to the next section.

7.2.2 Adding Graphics Drivers

To facilitate the adaptation to new graphics packages, COSY INFINITY has a very simple standardized graphics interface in the file FOXGRAF.FOP. In order to write drivers for a new graphics package, the user has to supply a set of seven routines interfacing to the graphics package. For ease of identification and uniformity, the names of the routines should begin with a three letter identifier for the graphics system, and should end with three letters identifying their task. The required routines are

1. ...BEG : Begins the picture. Allows calling all routines necessary to initiate a picture.
2. ...MOV(X,Y) : Performs a move of the pen to coordinates X,Y. Coordinates range from 0 to 1.
3. ...DRA(X,Y) : Performs a draw from the current position to coordinates X,Y. Coordinates range from 0 to 1.
4. ...CHA(I) : Prints ASCII character I to momentary position. Size of the character has to be 1/80 of the total picture size. After the character, the position is advanced in x direction by 1/80.
5. ...COL(I) : Sets a color. If supported by the system, the colors are referred to by the following integers: 1: black, 2: blue, 3: red, 4: yellow, 5: green.
6. ...WID(I) : Sets the width of the pen. I ranges from 1 to 10, 1 denoting the finest and 10 the thickest line.
7. ...END : Concludes the picture. Allows calling all routines necessary to close the picture and print it.

The arguments X and Y are DOUBLE PRECISION, and I is INTEGER. After these routines have been created, the routine GRPRI in FOXGRAF.FOP has to be modified to include calls to the above routines at positions where the other corresponding routines are called for other graphics standards.

We appreciate receiving drivers for other graphics systems written by users to include them into the master version of the code.

8 Acknowledgements

For discussions about languages and compilers, I want to thank Dr. Hiroshi Nishimura, Dipl.-Math. Dipl.-Phys. H. C. Hofmann, and Dipl.-Phys. Klaus Lindemann. I would like to thank Kurt Overley and Dr. Tom Mottershead for their optimization routine QOPT, and Jay Dittmann for help with the simplex optimizer.

I would like to thank Dr. Felix Marti for writing the GKS graphics drivers and helpful discussions about graphics. For testing different parts of the DA package, I would like to thank Dr. Etienne Forest, Dipl.- Phys. Bernd Hartmann and Dipl.-Phys. Stefan Meuser. I would also like to thank numerous COSY users for streamlining the implementations on other machines and many good suggestions.

Financial support was appreciated from the Deutsche Forschungsgemeinschaft, the University of Gießen, the Universities Research Association through the SSC Central Design Group, the Department of Energy through Los Alamos National Laboratory and Lawrence Berkeley Laboratory, and the National Science Foundation through Michigan State University and the National Superconducting Cyclotron Laboratory.

Finally, I would like to thank my wife for letting me go on all those evenings and weekends ...

References

[1] M. Berz. Computational aspects of design and simulation: COSY INFINITY. *Nuclear Instruments and Methods*, A298:473, 1990.

[2] M. Berz. COSY INFINITY, an arbitrary order general purpose optics code. *Computer Codes and the Linear Accelerator Community*, Los Alamos LA-11857-C:137, 1990.

[3] M. Berz. Direct computation and correction of chromaticities and parameter tune shifts in circular accelerators. *submitted to Physical Review A15*, 1991.

[4] M. Berz. Differential algebraic formulation of normal form theory. *submitted to Physical Review A15*, 1991.

[5] K. L. Brown. The ion optical program TRANSPORT. Technical Report 91, SLAC, 1979.

[6] T. Matsuo and H. Matsuda. Computer program TRIO for third order calculations of ion trajectories. *Mass Spectrometry*, 24, 1976.

[7] H. Wollnik, J. Brezina, and M. Berz. GIOS-BEAMTRACE, a computer code for the design of ion optical systems including linear or nonlinear space charge. *Nuclear Instruments and Methods*, A258:408, 1987.

[8] M. Berz, H. C. Hofmann, and H. Wollnik. COSY 5.0, the fifth order code for corpuscular optical systems. *Nuclear Instruments and Methods*, A258:402, 1987.

[9] M. Berz and H. Wollnik. The program HAMILTON for the analytic solution of the equations of motion in particle optical systems through fifth order. *Nuclear Instruments and Methods*, A258:364, 1987.

[10] M. Berz. Arbitrary order description of arbitrary particle optical systems. *Nuclear Instruments and Methods*, A298:426, 1990.

[11] M. Berz. Differential Algebraic description of beam dynamics to very high orders. *Particle Accelerators*, 24:109, 1989.

[12] M. Berz. Differential Algebraic treatment of beam dynamics to very high orders including applications to spacecharge. *AIP Conference Proceedings*, 177:275, 1988.

[13] M. Berz. *The Description of Particle Accelerators using High Order Perturbation Theory on Maps, in: M. Month (Ed), Physics of Particle Accelerators*, volume 1, page 961. American Institute of Physics, 1989.

[14] M. Berz. High-order description of accelerators using differential algebra and first applications to the SSC. In *Snowmass Summer Meeting*, Snowmass, Co, 1988.

[15] E. Forest and M. Berz. *Canonical Integration and Analysis of Periodic Maps using Non-Standard Analysis and Lie Methods*, page 47. Springer, Mexico City, 1989.

[16] M. Berz. Differential algebra precompiler version 3 - reference manual. Technical Report MSUCL-755, Michigan State University, East Lansing, MI 48824, 1990.

[17] E. Forest, M. Berz, and J. Irwin. Normal form methods for complicated periodic systems: A complete solution using Differential Algebra and Lie operators. *Particle Accelerators*, 24:91, 1989.

[18] H. Wollnik, B. Hartmann, and M. Berz. Principles behind GIOS and COSY. *AIP Conference Proceedings*, 177:74, 1988.

[19] B. Hartmann, M. Berz, and H. Wollnik. The computation of fringing fields using Differential Algebra. *Nuclear Instruments and Methods*, in print, 1989.

[20] B. Hartmann, H. Wollnik, and M.Berz. Tribo, a program to determine high-order properties of intense ion beams. *Computer Codes and the Linear Accelerator Community*, Los Alamos LA-11857-C:431, 1990.

[21] H. Wollnik, J. Brezina, and M. Berz. GIOS-BEAMTRACE, a program for the design of high resolution mass spectrometers. In *Proceedings AMCO-7*, page 679, Darmstadt, 1984.

[22] A. J. Dragt, L. M. Healy, F. Neri, and R. Ryne. MARYLIE 3.0 - a program for nonlinear analysis of accelerators and beamlines. *IEEE Transactions on Nuclear Science*, Ns-3,5:2311, 1985.

[23] C. Iselin and J. Niederer. The MAD program, version 7.2, user's reference manual. Technical Report CERN/LEP-TH/88-38, CERN, 1988.

[24] H. Wollnik. *Charged Particle Optics.* Academic Press, Orlando, Florida, 1987.

[25] Kasper and P. Hawkes. *Electron Optics.* Academic Press, Orlando, 1989.

[26] D. C. Carey. *The Optics of Charged Particle Beams.* Harwood, 1987.

[27] X. Jiye. *Aberration Theory in Electron and Ion Optics.* Advances in Electronics and Electron Physics, Supplement 17. Academic Press, Orlando, Florida, 1986.

[28] K. G. Steffen. *High Energy Beam Optics.* Wiley-Interscience, New York, 1965.

[29] W.Glaser. *Grundlagen der Elektronenoptik.* Springer, Wien, 1952.

[30] E.R. Cohen and B.N. Taylor. 1986 adjustments of the fundamental physical constants. *Review of Modern Physics*, 59:1121, 1987.

[31] A.J.Dragt and J.M.Finn. *Journal of Mathematical Physics*, 17:2215, 1976.

[32] A. J. Dragt. Lectures on nonlinear orbit dynamics. In *1981 Fermilab Summer School.* AIP Conference Proceedings Vol. 87, 1982.

[33] M. Berz. Isochronous beamlines for free electron lasers. *Nuclear Instruments and Methods*, A298:106, 1990.

9 Appendix: The Supported Types and Operations

The language of COSY INFINITY is object oriented, and it is very simple to create new data types and define operations on them. Details about the types and operations are described in the language description data file GENFOX.DAT which is read by the program GENFOX, which then updates the source code of the compiler.

The first entry in GENFOX.DAT is a list of the names of all data types. The second entry is a list containing the elementary operations, information for which combinations of data types they are allowed, and the names of individual FORTRAN routines to perform the specific operations.

The third entry contains all the intrinsic functions and the types of their results. The fourth entry finally contains a list of FORTRAN procedures that can be called from the environment.

All these data are read from a program that updates the compiler; in particular, it includes all the intrinsic operations, functions and procedures into the routine that interprets the intermediate code.

Below follows a list of all object types as well as a list of all the operands available for various combinations of objects, the available intrinsic functions, and the available intrinsic procedures. These lists are automatically produced in LaTeX format by the program GENFOX with each compiler update.

This information is current as of 13-JUN-91.

In this version, the following types are supported:

RE	8 Byte Real Number
ST	String
LO	Logical
DA	Differential Algebra Vector
VE	Real Number Vector
CM	8 Byte Complex Number
IN	8 Byte Interval Number
GR	Graphics
CD	Complex Differential Algebra Vector

Now follows a list of all operations available for various combinations of types. For each operation, a relative priority is given which determines the hierarchy of the operations in expressions if there are no parentheses.

- + (Addition) has priority 3 and is supported for the following combinations of types:

Left Type	Right Type	Type of Result
RE	RE	RE
RE	DA	DA
RE	VE	VE
RE	CM	CM
RE	IN	IN
RE	CD	CD
LO	LO	LO
DA	RE	DA
DA	DA	DA
DA	CM	CD
DA	CD	CD
VE	RE	VE
VE	VE	VE
CM	RE	CM
CM	DA	CD
CM	CM	CM
CM	CD	CD
IN	RE	IN
IN	IN	IN
CD	RE	CD
CD	DA	CD
CD	CM	CD
CD	CD	CD

- \- (Subtraction) has priority 3 and is supported for the following combinations of types:

Left Type	Right Type	Type of Result
RE	RE	RE
RE	DA	DA
RE	VE	VE
RE	CM	CM
RE	IN	IN
RE	CD	CD
DA	RE	DA
DA	DA	DA
DA	CM	CD
DA	CD	CD
VE	RE	VE
VE	VE	VE
CM	RE	CM
CM	DA	CD
CM	CM	CM
CM	CD	CD
IN	RE	IN
IN	IN	IN
CD	RE	CD
CD	DA	CD
CD	CM	CD
CD	CD	CD

- * (Multiplication) has priority 4 and is supported for the following combinations of types:

Left Type	Right Type	Type of Result
RE	RE	RE
RE	DA	DA
RE	VE	VE
RE	CM	CM
RE	IN	IN
RE	CD	CD
LO	LO	LO
DA	RE	DA
DA	DA	DA
DA	CM	CD
DA	CD	CD
VE	RE	VE
VE	VE	VE
CM	RE	CM
CM	DA	CD
CM	CM	CM
CM	CD	CD
IN	RE	IN
IN	IN	IN
CD	RE	CD
CD	DA	CD
CD	CM	CD
CD	CD	CD

- / (Division) has priority 4 and is supported for the following combinations of types:

Left Type	Right Type	Type of Result
RE	RE	RE
RE	DA	DA
RE	VE	VE
RE	CM	CM
RE	IN	IN
RE	CD	CD
DA	RE	DA
DA	DA	DA
DA	CM	CD
DA	CD	CD
VE	RE	VE
VE	VE	VE
CM	RE	CM
CM	DA	CD
CM	CM	CM
CM	CD	CD
IN	RE	IN
IN	IN	IN
CD	RE	CD
CD	DA	CD
CD	CM	CD
CD	CD	CD

- ^ (Exponentiation) has priority 5 and is supported for the following combinations of types:

Left Type	Right Type	Type of Result
RE	RE	RE

- < (Less Than) has priority 2 and is supported for the following combinations of types:

Left Type	Right Type	Type of Result
RE	RE	LO

- > (Greater Than) has priority 2 and is supported for the following combinations of types:

Left Type	Right Type	Type of Result
RE	RE	LO

- = (Equal) has priority 2 and is supported for the following combinations of types:

Left Type	Right Type	Type of Result
RE	RE	LO

- # (Not Equal) has priority 2 and is supported for the following combinations of types:

Left Type	Right Type	Type of Result
RE	RE	LO

- & (Concatenate) has priority 2 and is supported for the following combinations of types:

Left Type	Right Type	Type of Result
RE	RE	VE
RE	VE	VE
ST	ST	ST
VE	RE	VE
VE	VE	VE
GR	GR	GR

Now follows a list of all intrinsic functions available in the momentary version with a short description and the allowed types.

- TYPE returns the type of an object as a number and is supported for the following types:

Argument Type	Type of Result
RE	RE
ST	RE
LO	RE
DA	RE
VE	RE
CM	RE
IN	RE
GR	RE

- LENGTH returns the momentary length of a variable in 8 byte blocks and is supported for the following types:

Argument Type	Type of Result
RE	RE
ST	RE
LO	RE
DA	RE
VE	RE
CM	RE
IN	IN
GR	RE

- VARMEM returns the current memory address of an object and is supported for the following types:

Argument Type	Type of Result
RE	RE
ST	RE
LO	RE
DA	RE
VE	RE
CM	RE
GR	RE

- VARPOI returns the current pointer address of an object and is supported for the following types:

Argument Type	Type of Result
RE	RE
ST	RE
LO	RE
DA	RE
VE	RE
CM	RE
GR	RE

- EXP computes the exponential function and is supported for the following types:

Argument Type	Type of Result
RE	RE
DA	DA
VE	VE

- LOG computes the natural logarithm and is supported for the following types:

Argument Type	Type of Result
RE	RE
DA	DA
VE	VE

- SIN computes the sine and is supported for the following types:

Argument Type	Type of Result
RE	RE
DA	DA
VE	VE

- COS computes the cosine and is supported for the following types:

Argument Type	Type of Result
RE	RE
DA	DA
VE	VE

- TAN computes the tangent
  and is supported for the following types:

Argument Type	Type of Result
RE	RE
DA	DA
VE	VE

- ASIN computes the arc sine
  and is supported for the following types:

Argument Type	Type of Result
RE	RE
DA	DA
VE	VE

- ACOS computes the arc cosine
  and is supported for the following types:

Argument Type	Type of Result
RE	RE
DA	DA
VE	VE

- ATAN computes the arc tangent
  and is supported for the following types:

Argument Type	Type of Result
RE	RE
DA	DA
VE	VE

- SINH computes the hyperbolic sine
  and is supported for the following types:

Argument Type	Type of Result
RE	RE
DA	DA
VE	VE

- COSH computes the hyperbolic cosine
  and is supported for the following types:

Argument Type	Type of Result
RE	RE
DA	DA
VE	VE

- TANH computes the hyperbolic tangent
and is supported for the following types:

Argument Type	Type of Result
RE	RE
DA	DA
VE	VE

- SQRT computes the square root
and is supported for the following types:

Argument Type	Type of Result
RE	RE
DA	DA
VE	VE

- ISRT computes the reciprocal of square root
and is supported for the following types:

Argument Type	Type of Result
RE	RE
DA	DA
VE	VE

- SQR computes the square
and is supported for the following types:

Argument Type	Type of Result
RE	RE
DA	DA
VE	VE
CD	CD

- ABS computes the absolute value
and is supported for the following types:

Argument Type	Type of Result
RE	RE
DA	RE
VE	VE
CD	RE

- NORM computes the norm of a vector
and is supported for the following types:

Argument Type	Type of Result
DA	RE

- CONS determines the constant part
and is supported for the following types:

Argument Type	Type of Result
RE	RE
DA	RE
VE	RE
CM	CM
IN	RE
CD	CM

- REAL determines the real part
  and is supported for the following types:

Argument Type	Type of Result
RE	RE
DA	DA
CM	RE
CD	DA

- IMAG determines the imaginary part
  and is supported for the following types:

Argument Type	Type of Result
RE	RE
DA	DA
CM	RE
CD	DA

- INT determines the integer part
  and is supported for the following types:

Argument Type	Type of Result
RE	RE

- NINT determines the nearest integer
  and is supported for the following types:

Argument Type	Type of Result
RE	RE

- DA returns the i th elementary DA vector
  and is supported for the following types:

Argument Type	Type of Result
RE	DA

- CMPLX converts to complex
  and is supported for the following types:

Argument Type	Type of Result
RE	CM
DA	CD

- CONJ conjugates a complex number
  and is supported for the following types:

Argument Type	Type of Result
CM	CM
CD	CD

In addition to the just listed operators and intrinsic functions, the following intrinsic procedures are available:

- Procedure OPEN (3 arguments) opens a file. Parameters are unit number, filename (string), and status (string, same as in FORTRAN open). Necessary on some machines.
- Procedure CLOSE (1 argument) closes a file. Parameter is unit number.
- Procedure MEMALL (1 argument) returns the amount of memory allocated at this time
- Procedure MEMFRE (1 argument) returns the amount of memory still available at this time
- Procedure CPUSEC (1 argument) returns the elapsed CPU time in seconds since last midnight
- Procedure QUIT (1 argument) terminates execution; Argument = 1 triggers traceback
- Procedure SCRLEN (1 argument) sets the amount of space scratch variables are allocated with
- Procedure DAINI (4 arguments) initializes the order and number of variables of DA. Arguments are order, number of variables, output unit number, number of monomials (return).
- Procedure DANOT (1 argument) sets momentary truncation order for DA.
- Procedure DAEPS (1 argument) sets zero tolerance for components of DA vectors.
- Procedure DAREA (3 arguments) reads a DA vector. Arguments are the unit number, the variable name and the number of independent variables.
- Procedure DAPRV (5 arguments) prints an array of DA vectors. Arguments are the array, the number of components, maximum and current main variable number, and the unit number.
- Procedure DAREV (5 arguments) reads an array of DA vectors. Arguments are the array, the number of components, maximum and current main variable number, and the unit number.

- Procedure DAFLO (4 arguments) computes the flow of x' = f(x) for time step 1. Arguments: the initial condition, array of right hand sides, result, and dimension of f.

- Procedure DARAN (2 arguments) fills DA vector with random entries. Arguments are DA vector and fill factor.

- Procedure DADIU (3 arguments) performs a division by a unit vector if possible. Arguments are the number of the unit vector and the two DA vectors.

- Procedure DADER (3 arguments) performs the derivation operation on DA vector. Arguments are the number with respect to which to differentiate and the two DA vectors.

- Procedure DAINT (3 arguments) performs an integration of a DA vector. Arguments are the number with respect to which to integrate and the two DA vectors.

- Procedure DAPLU (4 arguments) replaces power of independent variable I by constant C. Arguments are the DA or CD vector, I, C, and the resulting DA or CD vector.

- Procedure DAPEE (3 arguments) returns a component of a DA vector. Arguments are the DA vector, the id for the coefficient in TRANSPORT notation, and the real component.

- Procedure DAPEP (4 arguments) returns a parameter dependent component of a DA vector. Arguments are the DA vector, the coefficient id, number of variables, and the result.

- Procedure DANOW (3 arguments) computes the order weighted norm of the DA vector in the first argument. Second argument: weight, third argument: result

- Procedure CDF2 (5 arguments) Lets exp(: f_2 :)) act on first argument in Floquet variables. Other Arguments: 3 tunes (2π), result

- Procedure CDNF (8 arguments) Lets $1/(1 - \exp(: f_2 :))$ act on first argument in Floquet variables. Other Arguments: 3 tunes (2π), array of resonances with dimensions, result

- Procedure CDNFDA (7 arguments) Lets $C_j^{\pm}$ act on the first argument. Other Arguments: moduli, arguments, coordinate number, total number, epsilon, and result

- Procedure MTREE (7 arguments) computes the tree representation of a DA array. Arguments: DA array, elements, coefficient array, 2 steering arrays, elements, length of tree.

- Procedure LINV (5 arguments) inverts a quadratic matrix. Arguments are the matrix, the inverse, the number of actual entries, the allocation dimension, and an error flag.
- Procedure MBLOCK (5 arguments) transforms a quadratic matrix to blocks on diagonal. Arguments are matrix, the transformation and its inverse, allocation and momentary dimension
- Procedure SUBSTR (4 arguments) computes a substring. Arguments are string, first and last numbers identifying substring, and substring.
- Procedure VELSET (3 arguments) sets a component of a vector of reals. Arguments are the vector, the number of the component, and the real value for the component to be set.
- Procedure VELGET (3 arguments) returns a component of a vector of reals. Arguments are the vector, the number of the component, and on return the real value of the component.
- Procedure VECELE (3 arguments) returns a component of a vector of reals. Arguments are the vector, the number of the component, and on return the real value of the component.
- Procedure VEZERO (3 arguments) used in repetitive tracking to prevent overflow due to lost particle.
- Procedure IMUNIT (1 argument) returns the imaginary unit i.
- Procedure TRUE (1 argument) returns the logical value true.
- Procedure FALSE (1 argument) returns the logical value false.
- Procedure INTERV (3 arguments) produces an interval from 2 numbers. Arguments are the lower and upper bounds and on return the resulting interval.
- Procedure INLO (2 arguments) returns the lower bound of an interval. Arguments are the interval and on return the lower bound.
- Procedure INUP (2 arguments) returns the upper bound of an interval. Arguments are the interval and on return the lower bound.
- Procedure INTPOL (2 arguments) determines coefficients of Polynomial satisfying $P(\pm 1) = \pm 1$, $P^{(i)}(\pm 1) = 0$, $i = 1, ..., n$. Arguments: coeffient array, n.
- Procedure CLEAR (1 argument) clears a graphics object
- Procedure GRMOVE (4 arguments) produces a graphics object containing just one move. Arguments are the three coordinates and the graphics object.

- Procedure GRDRAW (4 arguments) produces a graphics object containing just one draw. Arguments are the three coordinates and the graphics object.
- Procedure GRCOLR (2 arguments) produces a graphics object containing just one color change. Arguments are the new color id and the graphics object.
- Procedure GRWDTH (2 arguments) produces a graphics object containing just one width change. Arguments are the new width id and the graphics object.
- Procedure GRCHAR (2 arguments) produces a graphics object containing just one character. Arguments are the character and the graphics object.
- Procedure GRPROJ (3 arguments) produces a graphics object containing just one 3D projection identifier. Arguments are phi and theta in degrees and the graphics object.
- Procedure RKCO (5 arguments) sets the coefficient arrays used in the COSY eighth order Runge Kutta integrator.

Index