HLS Applications Library

C. Milardi

Introduction

The first level of the Control System for the DAΦNE project runs High Level Software (HLS) applications. They include all those programs which perform operations involving many machine elements like lattice tuning, closed orbit measurement and correction, feedback and machine modeling. All the large amount of software necessary to drive and get information from machine devices and diagnostics constitutes the Specific Device Software (SDS), running in the third level of the Control System. Examples of SDS applications are the routines which get information from a Synchrotron Radiation Monitor, measure the beam position from a set of four button pick-ups, and so on.

For HLS applications FORTRAN has been chosen as development language and the Macintosh computer as a support. The FORTRAN compiler by Language System [1] running under the MPW [2] development environment has been chosen. This compiler includes all the standard FORTRAN intrinsic functions; nevertheless it is necessary to provide a wide HLS Application Library (HAL) including mathematical tools as well as machine oriented routines. Such a library is the most efficient tool to support the efforts of those people who are involved in developing HLS applications. It optimizes the applications performance and reduces the amount of software stored in the control system consoles. Finally it makes the HLS maintenance and upgrade simpler.

The HAL is useful also in developing dedicated machine drivers; moreover if the SDS applications require the use of other languages, like C, it is possible to exploit the possibility of calling Fortran routines from C codes and vice versa provided by the compilers.

The HAL is the main utility that has to be provided to the Control System, since all the software tools like graphics, printing, file and data-base management are provided by the Control System user interface.

Concerning the organization of all input-output data dealing with the HLS applications, it has been decided to use the Real Time DataBase as a storing device. In this way the same lattice description, the same calibration and numeric constants will be available both to the HLS and the SDS applications.
1) HLS Application Library Basic Features

The software described in this paper is a second version of the HAL; the first one has been presented in the DAFNE Control System Status Report (Oct. 13th 1993).

There are some general criteria, which are mandatory in the interaction between the HLS applications and the Control System. These criteria consist in:

1) avoiding the use of COMMON variables.
2) avoiding the assignment of constants through the PARAMETER instruction.
3) avoiding the assignment of variables through the DATA instruction.
4) using array and matrix adjustable dimensioning as much as possible.
5) trying to generalize each routine.
6) fixing a standard formalism for the machine lattice description.
7) providing each routine with the proper escape sequence and output error code in case of fault.
8) providing each routine with a generous amount of comments.

Point 1 is fundamental for building routines of easy and general use. It is worth recalling that large amounts of data can be passed as a single actual argument exploiting the RECORD and STRUCTURE tools provided by the standard FORTRAN language.

The same generality requirements justify points 2 and 3. It is useful to emphasize that all the constants used in HLS application are stored in the Real Time DataBase of the Control System, where all the data for variables initialization are also available.

Arrays and matrices are extensively used in the HAL; in order to preserve generality it is better to use adjustable size objects. Where it is not permitted, like in the STRUCTURES, a proper, at least redundant dimensioning is recommended.

Point 5 means that whenever a routine is included in the HAL, it must be written taking into account all the possible applications it could be called from. This is very important to avoid the presence of almost equivalent routines in the same library.

All the accelerator physics routines need access to the machine layout and to the variables describing each machine element. This consideration makes point 6 a straightforward consequence of points 2, 3 and 5.

Mathematical calculations can sometimes give wrong results or end up with an unreasonable output. The HAL routines take into account such possibility, and have to provide the proper escape sequence together with an output error code to protect the Control System from infinite loops and inconsistent situations.
2) Communication between HLS Application and Control System

The communications between HLS and the Control System user interface are provided by the LVLibrary [3], which allows the HLS applications to access all the information stored in the Real Time DataBase and to send output data or commands to the Control System. This communication mechanism works well, but it may be time consuming if many accesses to the data base to collect information are required.

Nevertheless, if the above mentioned HAL general criteria are fulfilled, it becomes possible to exploit the LabView CIN utility. LabView [4] is the package used to develop the Control System; its programming language is not based on a sequence of instructions, but on block diagrams connecting nodes, which perform specific operations: tools and VI. The CIN [5] mechanism enables to assign an executable code written in a conventional language (C, FORTRAN, PASCAL) to a node.

In this way an HLS application uses the block diagram to get all the input data and to send back the output. This method could look definitely better that the previous one, however it has also some drawbacks: the execution of a CIN is synchronous, namely the application embedded in the CIN takes full control of the processor on which it is running. Any other event is ignored, unless the proper entries are provided inside the application. In order to create an executable application in the form of a CIN it is necessary to compile it using proper options and to make LabView and Fortran input-outputs compatible. It is definitely unrealistic that a Fortran expert ignoring LabView and C can create a CIN including an HLS application, while this can be easily done using the LVLibrary.

3) The Fortran Library

The HAL contains two different sections: mathematical routines and machine oriented routines.

3.1) Mathematical Section

It is well known that on main frames and workstations large and well funded mathematical libraries are available. At LNF the most popular are the Cern library, widely used by machine physicist, and the Nag Library, a well supported and documented high quality commercial product. A Macintosh version of the Nag library has been purchased, while contacts have been established to do the same for the CERN library. At the same time other products, specific for Macintosh, are being considered. Anyway the final library will provide at least matrix algebra, statistical functions, integration and differentiation utilities, special functions, resolution methods for differential equations, Fourier analysis, eigenvalue and eigenvector computation.

In the meantime a small number of basic mathematical routines has been included in the HAL, together with some routines, which are widely used in the existing Fortran codes developed by the Accelerator Physics Group members.
### 3.2) Machine Oriented Section

In section 1 I have stressed that the first step in developing machine oriented routines consists in setting a formalism to describe the machine lattice. Actually in Frascati three formalisms are used:

- **The MAD machine description**: it has been used in the design of the Transfer Lines and to find the optimum layout of orbit correctors and monitors, both in the Booster and in the Main Rings.

- **The LEDA input stream**: this code has been extensively used to design all the rings of the DAFNE project. Moreover, many other useful programs developed for the lattice design are based on this machine description; this is the case of the TRACK code [6] which optimizes the Booster injection-extraction efficiency and of the DAFNE code [7] which performs tracking with multipoles and dynamic aperture estimates.

- **The PARAMETER LIST machine description**: it has been created in order to provide a standard classification for the machine components, and it is useful to exchange data among different groups. It provides a very deep and complete machine description formalism.

This last remark has been the reason for the choice of the PARAMETER LIST formalism as the standard one for the HAL machine oriented routines [8]. Otherwise a translator would have been necessary, to make the MAD, LEDA and PARAMETER LIST formalisms compatible.

The STRUCTURE for the PARAMETER LIST input follows:

```plaintext
STRUCTURE / Element/
    integer*4 i,ident,itype,status
    character name*8
    real*8 length,plength,k2,fi,teta,b
    real*8 e1,e2,ro,ax,ay

END STRUCTURE

STRUCTURE / Lattice/
    integer*4 nelem,nperiod,symflag
    character title*8
    RECORD / Element/TheElement(1000)

END STRUCTURE

RECORD / Lattice/TheLattice

The Lattice structure contains an array substructure Element; it includes all the information on the machine optics:

- **TheLattice.nelem**: number of listed elements
  - if 0 then nelem is the total element number in a machine period.
  - If 1 then nelem is the total element number in a half machine period, reflection symmetry is assumed.
TheLattice.nperiod  
number of periods in the machine.

TheLattice.title  
the input file name or a comment, not longer than 8 characters.

TheLattice.TheElement.i  
progressive order of the listed element.

TheLattice.TheElement.ident  
element specific identifier, used to declare an element family.

TheLattice.TheElement.itype  
element identifier (see following table [9]).

TheLattice.TheElement.name  
element name.

TheLattice.TheElement.length  
element length (m).

TheLattice.TheElement.plength  
progressive total length (m).

TheLattice.TheElement.k2  
quadrupole K2 (m⁻²), horizontal dipole field index (m⁻²), sextupole K² (m⁻²), kicker deflection angle (rad).

TheLattice.TheElement.fi  
rotation angle (rad), used for skew quadrupoles.

TheLattice.TheElement.teta  
deflection angle (rad).

TheLattice.TheElement.B  
magnetic field (T).

TheLattice.TheElement.e1  
first pole face angle (rad), used for dipoles.

TheLattice.TheElement.e2  
second pole face angle (rad), used for dipoles.

TheLattice.TheElement.ro  
bending radius (m).

TheLattice.TheElement.Ax  
horizontal aperture (mm).

TheLattice.TheElement.Ay  
vertical aperture (mm).

TheLattice.TheElement.status  
element status: this information comes from the Control System.

<table>
<thead>
<tr>
<th>element</th>
<th>TheLattice.TheElement.itype</th>
</tr>
</thead>
<tbody>
<tr>
<td>DRIFT</td>
<td>1</td>
</tr>
<tr>
<td>QUADRUPOLE</td>
<td>2</td>
</tr>
<tr>
<td>SEXTUPOLE</td>
<td>3</td>
</tr>
<tr>
<td>HORIZONTAL DIPOLE</td>
<td>4</td>
</tr>
<tr>
<td>VERTICAL DIPOLE</td>
<td>44</td>
</tr>
<tr>
<td>VERTICAL CORRECTOR</td>
<td>5</td>
</tr>
<tr>
<td>HORIZONTAL CORRECTOR</td>
<td>6</td>
</tr>
<tr>
<td>HORIZONTAL&amp;VERTICAL CORRECTOR</td>
<td>56</td>
</tr>
<tr>
<td>SEPTUM</td>
<td>7</td>
</tr>
<tr>
<td>WIGGLER</td>
<td>8</td>
</tr>
<tr>
<td>RIGHT HALF-I.R. MATRIX</td>
<td>10</td>
</tr>
<tr>
<td>LEFT HALF-I.R. MATRIX</td>
<td>11</td>
</tr>
<tr>
<td>UNIFORM SOLENOID</td>
<td>14</td>
</tr>
<tr>
<td>OCTUPOLE</td>
<td>15</td>
</tr>
<tr>
<td>COORDINATE TRANSFORMATION</td>
<td>16</td>
</tr>
<tr>
<td>HORIZONTAL DISPERSION AND SLOPE</td>
<td>17</td>
</tr>
<tr>
<td>SYSTEMATIC MULTIPOLE</td>
<td>40</td>
</tr>
<tr>
<td>RANDOM MULTIPOLE</td>
<td>41</td>
</tr>
<tr>
<td>HORIZONTAL BPM</td>
<td>50</td>
</tr>
<tr>
<td>VERTICAL BPM</td>
<td>51</td>
</tr>
<tr>
<td>INJECTION&amp;EXTRACTION KICKER</td>
<td>70</td>
</tr>
<tr>
<td>DIAGNOSTICS (OTHER THEN BPM)</td>
<td>99</td>
</tr>
<tr>
<td>RF CAVITY</td>
<td>100</td>
</tr>
</tbody>
</table>
3.3) Routines Description

In the following the developed HAL routines are listed. Few of them have been taken from widely used programs or have been slightly modified; it is the case of the routines for matrix algebra and transport through accelerator elements. Others, like the "Nolisy" routine, have been rewritten in order to put them in general form. Finally, many of them have been written by myself from scratch in the course of the Accumulator design.

3.3.1) Mathematical Routines

**Subroutine MEq (a,r,n,m)**

Creates a matrix equal to an existing one.

**Input:**

- a(n,m) existing matrix.
- n number of rows.
- m number of columns

**Output:**

- r(n,m) output matrix.

**Subroutine MProd (a,b,r,n,m,l)**

Calculates the matrix product r of the a and b matrices.

**Input:**

- a(n,m) input matrix.
- n number of rows of the first matrix.
- m number of columns of the first matrix.
- b(m,l) input matrix.
- m number of rows of the second matrix.
- l number of columns of the second matrix.

**Output:**

- r(n,l) output matrix.

**Subroutine Idn (s,k)**

Returns the identity matrix s.

**Input:**

- k dimensions of the s matrix

**Output:**

- s(k,k) output matrix

**Subroutine Integral (vector,dx,nstep,result)**

Performs a step integration of the vector function.

**Input:**

- nstep number of integration steps.
- vector(nstep) values of the function to be integrated at each step.
- dx length of each step.

**Output:**

- result integral value.
Subroutine Trasp (sm,n,l,smt)
Computes the transpose of the sm matrix.
Input:
sm(n,l)  input matrix.
 n   number of rows.
 l   number of columns.
Output:
smt(l,n)  transpose matrix.

Subroutine Nolisy (TheLattice,TheCstNol,xx,cond,fct,fctj,ncall,TheFail)
Finds the numerical solution of the neq equations specified in the fct subroutine by using an improved Newton-Raphson method [10].
Input:
TheLattice record with the Lattice structure.
TheCstNol record with the CstNol structure.
STRUCTURE /TheNolCst/
   integer*4 neq,maxcall
   real*8 delta,eps,frac,fracmin,fracmax,p(10)
END STRUCTURE
RECORD /TheNolCst/NolCst
TheCstNol.neq number of equations.
TheCstNol.maxcall maximum number of calls allowed to the fct subroutine.
TheCstNol.eps accuracy of the solutions.
TheCstNol.delta numerical parameter.
TheCstNol.frac numerical parameter.
TheCstNol.fracmin numerical parameter.
TheCstNol.fracmax numerical parameter.
TheCstNol.p(10) numerical parameter.
xx (neq) initial values for the variables.
cond(10) searched solutions.
fct subroutine, declared as an external, which specifies the equations.
fctj subroutine declared as an external, which specifies the jacobian matrix of the equations to be solved.
Output:
xx(neq) solutions
ncall number of executed calls to the fct subroutine
TheFail record with the Fail structure.
TheFail.ifail 0 if satisfactory solutions found, otherwise an error code.
TheFail.comment explains extensively the occurred error.

Subroutine Inver (s,x,n,iflag)
Computes the inverse x of the nxn square matrix s.
Input
s(nxn)  input square matrix.
n   dimension of square matrix
Output
x  inverted matrix
iflag ≠ 0 if some error occurs in the inversion.
3.3.2) Machine oriented routines

SUBROUTINE ReadLattice (name,TheWLattice,js,jf,isym,np,alength,TheFail)
Reads the data stored in the TheWLattice record from the file called name. The file name is an EXCEL file providing free format data divided by a TAB character. If only a part of the data available is requested js jf give the progressive order of the elements at the beginning and at the end of the required range; in this case isym and np set the TheWLattice.isym and TheWLattice.np properties for the range selected; TheWLattice.nelem is updated by the routine.
Input:
name file name.
js progressive order of the first element, if a range is called, otherwise 0.
jf progressive order of the last element, if a range is requested, otherwise it is 0.
isym defines the TheWLattice.isym variable. It must be specified only if a range is selected.
np defines the TheWLattice.np variable. It must be specified only if a range is selected.
Output:
TheWLattice record with the Lattice structure.
alength total length of the magnetic structure described by the TheWLattice record.
TheFail record with the Fail structure.
TheFail.ifail = 0 if reading performed without problems; otherwise 1. An error occurs when: 1) a range is requested, but the starting file contains a only a partial description of the magnetic structure; 2) alength, obtained by summing up the single element lengths, is different from the machine progressive length referred to the last element stored in the TheWLattice record; 3) the total number of the input magnetic elements is different from the value stored in the TheWLattice.nelem field.
TheFail.comment specifies extensively the occurred error.

SUBROUTINE ExpLattice_SelChu (TheLattice,js,jf, isym, np,TheWLattice, alength, TheFail)
Reads the magnetic structure stored in the TheLattice record and if this is not given in an extended way (TheLattice.isym or TheLattice.np not 0) the routine expands the magnetic structure and stores the total configuration in the TheWLattice record. When only a part of the total machine is required js and jf give the progressive order of the elements limiting the required range; in this case isym and np set the TheWLattice.isym and TheWLattice.np properties for the selected range; TheWLattice.nelem is updated by the routine.
Input:
TheLattice record with the Lattice structure.
js progressive order of the range first element, if a range is requested, otherwise it is 0.
jf progressive order of the range last element, if a range is requested, otherwise it is 0.
isym value to assign to the TheWLattice.isym variable. It has to be specified only if a range is selected.
np value to assign to the TheWLattice.np variable. It has to be specified only if a range is selected.
Output:
TheWLattice record having the Lattice structure.
alength total length of the magnetic structure described by the TheWLattice record.
TheFail record having the Fail structure.
TheFail.ifail 0 if reading performed without problems; otherwise 1. An error occurs when the number of elements stored in the TheWLattice record is different from the TheWLattice.nelem value.
TheFail.comment specifies extensively the occurred error.

SUBROUTINE ZeroLKick (TheWLattice, alength, TheFail)
Fetches the magnetic structure stored in the TheLattice record and changes each injection/extraction kicker and sextupole into a localized kick in the center of the original one. The new magnetic configuration is stored in the TheWLattice record, the element lengths and the machine progressive lengths are updated accordingly.

Input:
TheLattice record with the Lattice structure.
Output:
TheWLattice record with the Lattice structure.
alength total length of the magnetic structure described by the TheWLattice record.
TheFail record with the Fail structure.
TheFail.ifail 0 if reading performed without problems; otherwise 1. An error occurs when alength, obtained by summing up the single element lengths, is different from the machine progressive length referred to the last element stored in the TheLattice record.
TheFail.comment specifies extensively the occurred error.

SUBROUTINE JoinLatChu (mp, ThePointStruc, isym, np, TheTLattice, alength, TheFail)
Joins mp machine chunks, each stored in a record with the Lattice structure. ThePointStruc array record provides the pointers to the mp records. The routine updates the TheTLattice.nelem value.

Input:
mp = number of machine chunks ≤30.
ThePointStruc (30) array record with the PointStruc structure
END STRUCTURE /PointStruc/
pointer /Lattice/ p
RECORD /PointStruc/ ThePointStruc (30)
isym value to assign to the TheTLattice.isym variable.
np value to assign to the TheWLattice.np variable.
Output:
alength total length of the magnetic structure described by the TheTLattice record.
TheTLattice record with the Lattice structure
TheFail record with the Fail structure.
TheFail.ifail 0 if reading performed without problems; otherwise 1. An error occurs when: 1) the total machine length, obtained by summing up the single element lengths for each chunk, is different from the machine progressive length referred to the last element stored in the TheLattice record; 2) the total number of the magnetic elements, obtained as a sum of the single chunk elements, is different from the total number of elements listed in the TheTLattice record.
NOTE

The address of each chunk record can be assigned to the pointer in each component of the `ThePointStruc` array record by using the following instructions:
ThePointStruc(1).p = %loc(The1Lattice).
ThePointStruc(2).p = %loc(The2Lattice).
ThePointStruc(3).p = %loc(The3Lattice).

Subroutine MatD (elle,s)
Gives the transport matrix $S$ for a straight section.

Input:

elle straight section length (m).

Output:

s (5,5) transport matrix.

Subroutine MatB (elle,b,rag,de,s)
Gives the transport matrix $S$ for a bending magnet, energy dependent effects are taken into account if $de \neq 0$.

Input:

elle magnet length (m).
b magnet field index.
rag magnet bending radius (m).
de relative energy deviation.

Output:

s (5,5) transport matrix.

Subroutine MatQ (elle,qk,de,s)
Gives the transport matrix $S$ for a quadrupole, energy dependent effects are taken into account if $de \neq 0$.

Input:

elle quadrupole length (m).
qk quadrupole $K^2$ (m$^{-2}$).
de relative energy deviation.

Output:

s (5,5) transport matrix.

Subroutine SexKick (sextp,xx)
Describes the effect of a sextupole in the localized angular perturbation approximation.

Input:

sextp $\left(1/Bro\right) \left(\partial^2 B/\partial x^2\right)$ (m$^{-2}$).
xx (5) particle coordinate array before the sextupole kick; xx(1) = horizontal position (m), xx(2) = horizontal coordinate derivative with respect to the longitudinal one (rad), xx(3) = energy deviation, xx(4) = vertical position (m), xx(5) = vertical position coordinate derivative with respect to the longitudinal one (rad)

Output:

xx (5) particle coordinate array after the sextupole kick.
Subroutine Stability (x,xmx,TheFail)
Checks both radial and vertical coordinate absolute value verifying they don’t exceed xmx.
Input:
x (5) coordinates.
xmx maximum displacement.
Output:
TheFail record with the Fail structure.
TheFail.ifail 0 if the condition is respected otherwise 1.
TheFail.comment explains extensively the occurred error.

Subroutine IEKicker (xkick,xx,r,v)
Describes the effect of a kicker in the localized angular perturbation approximation.
If r and v are not 0 xkick is calculated in the model of four infinite current wires, parallel to each other and to the straight section axis. In this case r and v are the horizontal and vertical distances between the wires.
Input:
xkick localized angular deflection (rad).
xx (5) particle coordinate array before the kick.
r horizontal distance among wires in the four currents kicker model (m).
v vertical distance among wires in the four currents kicker model (m).
Output:
xx (5) particle coordinate array after the sextupolar kick.

Subroutine TTE (TheElement,s)
Gives the transport matrix s through a magnetic element.
Input:
TheElement record with the Element structure, which describes the specific machine element.
Output:
s (5,5) transport matrix.

Subroutine TTM (TheLattice,s,TheFail)
Gives the total transport matrix s over the machine period.
Input:
TheLattice record with the Lattice structure describing the machine.
Output:
s [5,5] total transport matrix over the machine period.
TheFail record with the Fail structure.
STRUCTURE / Fail/
   integer*4 ifail
   character*80 comment
END STRUCTURE
0 if s (1,1) + s (2,2) + s (3,3) + s (4,4) ≤ 2
TheFail.ifail =
1 if s (1,1) + s (2,2) + s (3,3) + s (4,4) > 2, unstable machine
TheFail.comment explains extensively the occurred error.
**Subroutine TPM (tra,tss)**
Gives the transport matrix \( tss \) for the Twiss parameters using the element transport matrix \( tra \).

**Input:**
- \( tra \) (5,5) element transport matrix.

**Output:**
- \( tss \) (6,6) Twiss parameter transport matrix.

**Subroutine TwissP (tra,tp)**
Gives the Twiss parameters \( tp \) at the starting element of the lattice deck using the machine total transport matrix \( tra \).

**Input:**
- \( tra \) (5,5) machine total transport matrix.

**Output:**
- \( tp \) (6) Twiss parameters:
  - \( tp(1) = bx(m) \)
  - \( tp(2) = ax \)
  - \( tp(3) = gx(m^{-1}) \)
  - \( tp(4) = by(m) \)
  - \( tp(5) = ay \)
  - \( tp(6) = gy(m^{-1}) \).

**Subroutine BPhaseTM (TheLattice,tp,px,pz,l,m)**
Gives the betatron phase advances \( px \) and \( pz \) using the transport matrix of the machine elements between progressive numbers \( l \) and \( m \).

**Input:**
- \( TheLattice \) record with \( Lattice \) structure.
  - \( l \) progressive number of the first element.
  - \( m \) progressive number of the last element.
  - \( tp \) (6) Twiss parameters at the \( l \)-th element.
  - \( px \) horizontal betatron phase at the \( l \)-th element.
  - \( pz \) vertical betatron phase at the \( l \)-th element.

**Output:**
- \( tp \) (6) Twiss parameters at the \( m \)-th element.
  - \( px \) horizontal betatron phase at the \( m \)-th element.
  - \( pz \) vertical betatron phase at the \( m \)-th element.

**Subroutine BPhasel (TheLattice,tp,px,pz,l,m)**
Gives the machine betatron phase advances \( px \) and \( pz \) by integrating the betatron functions over the range selected by the elements between progressive numbers \( l \) and \( m \).

**Input:**
- \( TheLattice \) record with the \( Lattice \) structure.
  - \( l \) progressive number of the first element.
  - \( m \) progressive number of the last element.
  - \( tp \) (6) Twiss parameters at the \( l \)-th element.
  - \( px \) horizontal betatron phase at the \( l \)-th element.
  - \( pz \) vertical betatron phase at the \( l \)-th element.

**Output:**
- \( tp \) (6) Twiss parameters at the \( m \)-th element.
  - \( px \) horizontal betatron phase at the \( m \)-th element.
  - \( pz \) vertical betatron phase at the \( m \)-th element.

**Subroutine Tunes (TheLattice,tp,qx,qz)**
Gives the total tunes \( qx \) and \( qz \) by integrating the betatron function.

**Input:**
- \( TheLattice \) record with the \( Lattice \) structure.
  - \( tp \) (6) Twiss parameters at the starting point of the ring description.
Subroutine Dispersion (tra, eta)
Calculates the dispersion function eta at the ends of a machine period from the transport matrix over the period.

Input:
tra (5,5) transport matrix through a machine period
Output:
eta (5) dispersion function
eta(1) = h_x; eta(2) = h'_x; eta(3) = 1;
eta(4) = h_y; eta(5) = h'_y.

Subroutine TMCo (TheLattice, nv, TheCstMat)
Represents the machine in the form of a sequence of elements, marked by an identifier listed in the nv array, and constant matrices. The last ones are obtained by multiplying the transfer matrices of all those elements between two having an identifier listed in the nv array.

Input:
TheLattice record with the Lattice structure.
nv (599) elements identifiers.
Output:
TheCM(600) array record with the CM structure.
TheCstMat record with the CstMat structure.

STRUCTURE /CM/
real*8 co(5,5)
END STRUCTURE

STRUCTURE /CstMat/
integer*4 nt, ntc, ncs(1199)
RECORD /CM/TheCM(600)
END STRUCTURE

nt number of steps into which the machine is divided.
ntc total number of constant matrices.
ncs(1199) array with nt meaningful elements, each being either the progressive number of an element in the TheLattice record, or 0 for a constant step.
co(5,5) transport matrix for each constant step.

Subroutine CoMat (sm, TheCstMat)
Stores the sm matrix in the proper co matrix inside the TheCstMat record.

Input:
TheCstMat record with the CstMat structure.
sm (5,5) transfer matrix for a machine chunk.
Output:
TheCstMat record having the CstMat structure.

Subroutine TTCM (TheCstMat, TheLattice, tra, TheFail)
Gives the total transport matrix tra using data stored in the TheCstMat record.

Input:
TheLattice record with Lattice structure.
TheCstMat record with the CstMat structure.
Output:
tra (5,5) machine total transport matrix.
TheFail record with the Fail structure.
TheFail.ifail If 0 stable machine, if 1 unstable machine. The check is performed on the tra matrix.
TheFail.comment explains extensively the occurred error.

Subroutine TrackCM (xx,nturn,i,j,TheCstMat,TheLattice,ctr,TheFail)
Tracks a particle with initial coordinates xx through a machine described by the TheLattice record for nturn turns. It uses the machine chunk representation provided by the TheCstMat record. If nturn = 1 tracking can be performed between chunks having progressive numbers i and j.

Input:
xx particle initial coordinates.
nturn number of turns to be tracked.
i progressive number of the initial chunk (only if nturn = 1).
j progressive number of the last chunk (only if nturn = 1).
TheCstMat record with the CstMat structure.
TheLattice record with Lattice structure.
ctr maximum displacement of the horizontal position in sextupoles.

Output:
xx particle coordinates after nturn turns.
TheFail record with the Fail structure.
TheFail.ifail 1 if the radial coordinate xx(1) exceeds ctr in a sextupole.
TheFail.comment explains extensively the occurred error.

Subroutine RspMat (TheLattice,tijh,tijv,nmon,ncorh,ncorv,qx,qz)
Calculates the response matrix between each one of the nmon monitors and the ncor correctors, for both horizontal and vertical planes.

Input:
TheLattice record with the Lattice structure.

Output:
nmon number of monitors in the machine.
ncorh number of horizontal correctors.
ncorv number of vertical corrector.
tijh(nmon,ncorh) horizontal response matrix.
tijv(nmon,ncorv) vertical response matrix.
qx horizontal betatron tune.
qz vertical betatron tune.

4) Programs Description

The HAL Library has been used and debugged by developing some basic programs for the DAFNE Accumulator. These programs are listed in the following.

PROGRAM AccIKick
Calculates the strength (analytical computation) of the injection kickers for the DAFNE Accumulator in the ideal approximation of a symmetric orbit deformation. It uses a chunk representation of the machine with fixed matrices interleaved with sextupoles and injection kickers. The considered fraction of the ring magnetic configuration begins at the injection septum and ends up at the second injection kicker.
**Program AccEKick**

Calculates the strength of the extraction kickers for the DAFNE Accumulator. As a first step the program finds the strength of the most effective couple of extraction kickers (those near to the extraction septum) and then tries to decrease the required angular kicks by using also the other two; a check is performed, controlling that with the optimized solution the beam does not scrape the injection septum. The program uses a chunk representation of the machine with fixed matrices interleaved with sextupoles and injection kickers.

**Program AccSetnu**

Gives the quadrupole strengths of the DAFNE Accumulator required for given horizontal and vertical betatron tunes. Such strengths are obtained numerically for vanishing dispersion at the injection septum.

**4.1) Specific Routines**

In the following the Specific Routines used in the previous programs are described.

**Subroutine FctTunAcc (TheLattice,xs,cond,fx,TheFail)**

Specifies the equations to be solved in order to set the betatron tunes of the Accumulator. This routine is called within the AccSetnu program.

*Input:* 
- TheLattice: record with Lattice structure describing the Accumulator.
- fx(3): f(1) = horizontal tune, f(2) = vertical tune, f(3) = dispersion at the injection septum.
- cond(50): solutions requested for the equations fixed in this routine.

*Output:* 
- xs(3): quadrupole strengths.
- TheFail: record with the Fail structure.
- TheFail.ifail: 1 if the transport matrix for the whole machine is unstable.
- TheFail.comment: explains extensively the occurred error.

**Subroutine FctJTunAcc (TheLattice,xs,fr,cond,TheCstNol,dj,TheFail)**

Computes the Jacobian determinant \( dj \) of the \( fx \) function defined in the FctTunAcc routine. This routine is called from the AccSetnu program.

*Input:* 
- TheLattice: record with Lattice structure.
- xs: quadrupole strengths.
- fr: difference between the \( fx \), as specified in the FctTunAcc, computed using the initial \( xs \) value at each numerical iteration of the Nolisy routine.
- TheCstNol: record with the CstNol structure.
- cond(50): solutions requested for the equations to be solved.

*Output:* 
- \( dj(50,51) \): Jacobian determinant
- TheFail: record with the Fail structure.
- TheFail.ifail: 1 if the transport matrix for the whole machine is unstable.
- TheFail.comment: explains extensively the occurred error.
Subroutine IKick \((x_0, xp_0, \text{TheLattice}, \text{TheCstMat}, x_{ki})\)

Gives the strength (analytical computation) of the injection kicker \(x_{ki}\) for the DAFNE Accumulator. \(\text{TheCstMat}\) contains a chunk representation of the machine with fixed matrices interleaved with sextupoles and injection kickers. \(\text{TheLattice}\) describes the optics starting from the injection septum up to the second injection kicker included. This routine is called from the \(\text{AcclKick}\) program.

Input:
- \(x_0\): required stored beam trajectory displacement at straight section center.
- \(xp_0\): required stored beam trajectory slope at straight section center.
- \(\text{TheLattice}\): record with \(\text{Lattice}\) structure.
- \(\text{TheCstMat}\): record with \(\text{CstMat}\) structure.

Output:
- \(x_{ki}(2)\): injection kicker strength (rad).

Subroutine EKick \((x, xp, xx, \text{TheLattice}, \text{TheCstMat}, x_{ke})\)

Gives the strengths (analytical computation) of most effective extraction kickers \(x_{ke}\) for the DAFNE Accumulator. These are obtained from the required coordinates \(x, xp\) at the septum and the intermediate coordinates \(xx\) given by the minimization procedure of the \(\text{AccEKick}\) program. The subroutine uses a chunk representation of the machine with fixed matrices interleaved with sextupoles and injection kickers, provided by the \(\text{TheCstmat}\) record.

Input:
- \(x\): required beam trajectory position at the extraction septum.
- \(xp\): required beam trajectory slope at the extraction septum.
- \(xx(5)\): beam position and slope before most effective kickers.
- \(\text{TheLattice}\): record with the \(\text{Lattice}\) structure.
- \(\text{TheCstMat}\): record with the \(\text{CstMat}\) structure.

Output:
- \(x_{ke}(2)\): extraction kicker strength (rad).

5) Remarks

The software described is available in the \(\text{Catia}\) folder on the \(\text{Utenti}\) disk accessible from the \(\text{LNF Div.Acc.}\) file server, together with the specific instructions to compile link and run it in the MPW environment.

The listed routines can be found in the \(\text{HALibrary.f}\) file, while all the used structures are in the \(\text{HADataType.h}\) file, which has to be included in every main program calling the HALibrary routines, finally the \(\text{HLSinstall}\) folder provides the necessary MPW procedures.

In the following the steps necessary in order to write an HLS application using the HALibrary routines are listed:

1) Copy the \(\text{UserStartup•HLS}\) file inside the \(\text{MPW}\) folder and restart MPW.
2) Create a new folder called \(\text{HLS}\) inside the \(\text{MPW}\) folder.
3) Copy the \(\text{scripts}\) folder inside the \(\text{HLS}\) folder.
4) Copy the **HADDataType.h** file inside the **MPW:Libraries:Flibraries** folder.

5) Copy the **HALibrary.f** file inside the **HLS** folder and compile it with the instruction `fortran HALibrary.f`. The expert user can specify some compiling options, according to the processors available on his Macintosh, in order to optimize the routines performances.

6) Copy the **HALibrary.f.o** file inside the **MPW:Libraries:Flibraries** folder.

7) Develop your own application code including the **HLSData:HADDataType.h** file and compile, link and run it using the command listed in the scripts folder.

---

**6) References**


[5] CIN.


