1 Introduction

This note is the documentation for a Mathematica [1,2] package TrackTable designed to manipulate particle tracking data generated by the MAD program [3]. Functions contained in this package allow you to generate a TrackTable data object and extract data from it as well as transform it into some other useful forms (preserving the TrackTable type and all the other information saved in MAD’s TRACK table).

The package has been thoroughly tested with tables from MAD Version 8 and should also work with the tables from MAD Version 9.

The package includes functions for making canonical transformations to the eigenvector representation or the action-angle variables in 6-dimensional phase space, computing the decay of intensity of an ensemble of particles (e.g. to simulate a kicked-beam measurement), a function to “strobe” data to look for resonant behaviour and some algebra of symplectic matrices.

Recent additions to the package (early 1999) include a new data SurvivalTable data object for information about the survival of particles (derived from TrackTable objects). Thanks to natural inheritance, many of the functions acting on TrackTable objects have been straightforwardly overloaded to act on SurvivalTable objects.

Another new feature is a palette of buttons that free the user from most of the need to remember and type in the syntax of the functions in the package. See Section 9 below.

Access to the packages has been simplified thanks to changes in the CERN computer systems and “User’s Guide and Examples” below have been enhanced.

It is very likely that further functions will be added in future versions of the package. Therefore it is always worth checking the latest version of this notebook at the Madtomma Web site

http://wwwslap.cern.ch/jowett/Madtomma

Simple and convenient access to the data in the Mathematica environment opens up an enormous range of possibilities for analysing (or “post-processing”) the results of tracking with MAD. A few simple examples are given in the Examples section below.
Since MAD itself can be run from Mathematica it is also easy to build high-level interactive or batch interfaces to MAD from Mathematica notebooks, e.g., to explore phase space interactively by adjusting initial conditions for tracking according to criteria arrived at by some analysis of tracking data that would be hard to implement in the MAD language.

The data should be initially read in as (or transformed into) a basic mfs data object using the function tfsRead (from the package Mfs [4] which is automatically loaded by the present package). To simplify the usage, the package extends the operation of functions on the basic Mfs objects to deal with TrackTable data objects in natural ways. This keeps the set of defined functions small and makes life easier for the user.

These packages and others relating to the use of MAD will all be made available in the context Madtomma (the word context is used in a special sense in Mathematica). Preliminary versions of some are already available; documentation will be released as it is prepared.

This notebook adheres to the conventions for Mathematica package structure and documentation set out in [2]. As such it serves as the development medium for the package itself. For this reason, some sections of the printed versions of this document are hidden. They contain the definitions ("code") of the functions. The visible sections contain the documentation and examples of interest to users.

To make the package available, you need only copy a couple of commands from the "Setup" section. The "User's Guide and Examples" section illustrates basic use of the package. A final subsection illustrating some less useful features is hidden.

This notebook is based on the template for package and notebook development by R. Maeder and supplied with Mathematica 3.0. If you are reading this document in the Mathematica Front End, you will see that it contains hypertext links within itself and to pages on the World-Wide Web.

1.1 On programming paradigms

The Mathematica language supports most programming methodologies although functional programming is probably the most natural and powerful. However, since object-oriented programming (OOP) is currently popular and familiar to many, it is worth describing the present implementation in its terms. In fact, the functional programming paradigm, combined with pattern-matching, provides all the advantages of OOP - and much more besides - with very little effort.

In the terminology of OOP, the functions in this package can be thought of as methods for the TrackTable and SurvivalTable data object data objects, just as the functions in the Mfs package [4] were methods for the mfs objects. In fact, TrackTable objects (and others belonging to the same class) are a subtype of the more general mfs objects and therefore acquire many of their properties by inheritance. This is reflected by the fact that functions in the Mfs package are overloaded (by pattern-matching on their arguments) to deal with TrackTable objects. A call to a function corresponds to message-passing. Thus some of the basic functions of the Mfs package (e.g. mfsColumn) become polymorphic in this package.

2 Reference

2.0.1 Title

Tracking data from MAD
2.0.2 Authors
Krzysztof Goral, John M. Jowett

2.0.3 Summary
This notebook provides functions to read and manipulate tracking data saved in files by the program MAD. These tables are usually saved using the command

ARCHIVE,table="TRACK",filename="somefile.tfs"

2.0.4 Copyright
© Copyright CERN.1997

The copyright and all other rights relating to this computer software, in whatever form, including but not limited to the source code, the object code and user documentation, are vested in CERN.

CERN, on a royalty-free and non-exclusive basis, hereby grants permission to use, copy, change, modify, translate, display, distribute and make available this computer software, subject to the following conditions:

(1) this computer software is provided on an as-is basis and CERN provides no express or implied warranties of any kind, including but not limited to those of merchantability, fitness for a particular purpose and non-infringement of the proprietary rights, such as copyrights, patents and trade secrets, of third parties. CERN accepts no liability whatsoever for or in connection with the use of this computer software;

(2) all copies made of this computer software or of parts thereof shall include this copyright statement in full;

(3) however, if this computer software or parts thereof are made available in any other form than their original form, or are included in any other computer software, the following short acknowledgement only must be mentioned in the copyright statement and in the user documentation (or, in the absence thereof, in any other appropriate place) concerning the computer software thus made available or created:

"This product includes computer software created and made available by CERN. This acknowledgement shall be mentioned in full in any product which includes the CERN computer software included herein or parts thereof."

2.0.5 Notebook Version
3.2

2.0.6 Mathematica Version
3.0 (but should also work with earlier versions)

2.0.7 History
Version 1.0, KG & JMJ 29 August 1997

Version 1.1, 7 April 1998, JMJ bug in toActionAngle fixed, function made more compact.
Version 2.0, 21 January 1999, JMJ addition of palette, SurvivalTable data and functions.
Version 3.0, 14 September 1999, JMJ made compatible with Version 2.0 of mfs package.
Version 3.1, 22 March 2000, JMJ importing package for algebra of symplectic matrices, so removing functions from this package. The function \texttt{symplecticJ} changed its name to \texttt{SymplecticJ}.
Version 3.2, 16 August 2000 JMJ added phaseSpectrum

\subsection*{2.0.8 Keywords}
Track, dynamics, TFS, MAD, data, table, optics, post-processing, power spectrum

\subsection*{2.0.9 Warnings}
Note: all cells marked as "InitializationCell" will be written to the Auto-Save package. This package can then be read in programs that use it with \texttt{Needs["Madtomma'Mfs'Mfs'"]}. Cells not intended to belong to the package should not have this property.

Present versions of MAD Version 8 do not include the last two components of the closed orbit in the header block of the TRACK table. This results in error messages when TRACK tables are read in. This package anticipates the correction of this problem but will work without it.

\subsection*{2.0.10 Discussion}
Here we provide the internal specification of \texttt{TrackTable} data objects. Most users may skip it.

A TRACK table file from MAD can be read directly into an \texttt{Mfs} data object [4]. However this representation is not very convenient as the main data block starts with the coordinates of all particles on the first turn, followed by those of all surviving particles on the second turn and so on. As particles are lost, the number of rows corresponding to each turn diminishes. The main purpose of the function \texttt{makeTrackTable} is to transform this flat 2-dimensional list into a 3-dimensional list in which coordinates of different particles are separated.

A \texttt{TrackTable} data object consists of three parts. The first two of them are similar to those of the corresponding \texttt{mfs} object [4]. The first one is a list of descriptors and their values. The second one provides the labels for the data columns grouped in the third part. The third part contain the main data block and is in the form of a nested list. Lists of the first level correspond to turns done by particles in the storage ring whereas the second level is indexed by the particles.

\texttt{Mfs} functions operating on the first two parts of a \texttt{TrackTable} data object require only a slight redefinition as the structure is preserved. For the main data block the structure changes substantially and the functions manipulating it have a new shape.

\subsection*{2.0.11 Requirements}
Uses a basic package for reading TFS data files and creating \texttt{mfs} data objects.

\texttt{Madtomma'Mfs'Mfs} [4]
This in turn uses one of the Standard Packages (supplied with \texttt{Mathematica}) for column manipulations, etc.

\texttt{Statistics'DataManipulation}'
Imports a standard package for handling physical units.

\texttt{Miscellaneous'SIUnits'}
3 Setup

This section contains commands needed to load the corresponding package file. The contents of this file are equivalent to the following sections (Interface, Implementation, Epilog) in which the package is developed.

3.1 Search Path (ESSENTIAL!)

To have access to my packages, you may need to add my packages directory to your search path. This is system-dependent and the latest information about arranging it on CERN computer systems can be found at

http://wwwslap.cern.ch/~jowett/Madtomma/AboutFiles.html

and is not reproduced here. I strongly recommend that you modify your kernel initialisation file once and for all as explained on this page. Then all my packages will be found as easily as the Standard Packages that come with Mathematica.

3.2 Loading the Package

Once the package directory is on your search path, the \texttt{Mfs} package can be loaded by evaluating the following cell.

\begin{verbatim}
Needs"Madtomma\Mfs\TrackTable"
\end{verbatim}

Version 2.0 of Madtomma\Mfs\Mfs\TrackTable loaded.
Note function name changes since Version 1.x:

\begin{itemize}
  \item interpretDescriptors \rightarrow \texttt{mfsInterpretKeys}
  \item keyValue \rightarrow \texttt{mfsKeyValue}
  \item columnNames \rightarrow \texttt{mfsColumnNames}
  \item descriptorNames \rightarrow \texttt{mfsKeyNames}
  \item colValue \rightarrow \texttt{mfsColumnValue}
  \item addDescriptor \rightarrow \texttt{mfsAddKey}
  \item editDescriptor \rightarrow \texttt{mfsEditKey}
  \item deleteDescriptor \rightarrow \texttt{mfsDeleteKey}
  \item symplecticJ \rightarrow \texttt{SymplecticJ}
\end{itemize}

Version 3.1 of Madtomma\Mfs\TrackTable loaded. Name change for symplecticJ \rightarrow \texttt{SymplecticJ}

This is all you need to start using the package in your own applications. In interactive sessions, you will normally see a palette of buttons appearing on your screen. This makes it easier to use the package: see Section 9 below.

These following sections are hidden when this notebook is used as the package documentation but may be inspected in the online copy that can be found in the appropriate sub-directory of the directory added to the \texttt{\$Path} variable above.
4 Interface
5 Implementation
6 Epilog
7 Basic documentation for TrackTable package
8 Basic documentation for Mfs package

The TrackTable package loads and is based on the basic Mfs package so it may be convenient to generate its usage messages here.

```
UsageCells[Names["Madtomma\'Mfs\'Mfs\'\"]]
```

8.1 addDescriptor
Obsolete function name. Please use mfsAddKey.

8.2 columnNames
Obsolete function name. Please use mfsColumnNames.

8.3 colValue
Obsolete function name. Please use mfsColumnValue.

8.4 deleteDescriptor
Obsolete function name. Please use mfsDeleteKey.

8.5 descriptorNames
Obsolete function name. Please use mfsKeyName.

8.6 editDescriptor
Obsolete function name. Please use mfsEditKey.

8.7 interpretDescriptors
Obsolete function name. Please use mfsInterpretKeys.

8.8 interpretTagValue
interpretTagValue[{"tag",val}] creates a variable tag and assigns it the value val.

8.9 keepTemporaryFile
keepTemporaryFile is an option for tfSRead that causes a temporary file not to be deleted.

8.10 keyValue
Obsolete function name. Please use mfsKeyValue.
8.11 matchColumns

matchColumns is an option for mfsMerge that gives a list of column names that must match in all the mfs objects in which they appear. The value Automatic insists that all possible matches hold.

8.12 mfs

mfs is a data object type containing a set of scalar variables with values and a set of labelled columns of data; typically it can represent the data structures in TFS files.

8.13 Mfs

Mfs.m is a package for manipulating MFS data.

8.14 mfsAddColumn

mfsAddColumn[qp,"colname",coldata] returns an mfs object equal to qp but with a new column labelled colname and containing data coldata added. A list of column names and columns of data can also be given.

8.15 mfsAddKey

mfsAddKey[mfsdata,{"KEY",value}] returns the mfs (or related) data object mfsdata with an additional key and value.

8.16 mfsColumn

mfsColumn[mfsdata,colname] returns the column of data labelled by the string colname in an mfs (or related) data object. A list of colnames may also be given to return a set of columns. If colname is absent the entire block of columns is returned.

mfsColumn[TrackTabledata,n,colname] extracts a list of coordinates of particle n labelled by the string colname from a TrackTable data object. A list of colnames may also be given to return a set of lists. If colname is absent the entire block of coordinates is returned. If the particle number is absent a list of the requested column for all the particles is returned.

mfsColumn[SurvivalData,colname], mfsColumn[SurvivalData,particle,colname], mfsColumn[SurvivalData] are the obvious extensions to SurvivalTable objects.

8.17 mfsColumnMatch

mfsColumnMatch[{qp1,qp2,...},{"col1","col2",...}] tests whether a list of column names match in all the mfs objects in which they appear.

8.18 mfsColumnNames

mfsColumnNames[mfsdata] returns the list of column names in an mfs data (or related) object.

8.19 mfsColumnValue

mfsColumnValue[mfsdata,row,key] extracts a value labelled by key in row from an mfsdata object.

8.20 mfsDeleteColumn

mfsDeleteColumn[qp,"colname"] returns an mfs object equal to qp but with the column labelled colname removed. A list of column names can also be given.
8.21 mfsDeleteKey
mfsDeleteKey[mfsdata,key] returns the mfs (or related) data object mfsdata with the descriptor key removed.

8.22 mfsEditKey
mfsEditKey[mfsdata,{key,newValue}] returns the mfs (or related) data object mfsdata with the value corresponding to the descriptor key replaced by newValue.

8.23 mfsInterpretKeys
mfsInterpretKeys[mfsdata] creates a set of variables with values from the keys in an mfs (or related) data object.

8.24 mfsKeyNames
mfsKeyNames[mfsdata] returns the list of key names in the header block of an mfs (or related) data object.

8.25 mfsKeyValue
mfsKeyValue[mfsdata,key] returns the value corresponding to a descriptor key (a string) in an mfs (or related) data object.

8.26 mfsMember
mfsMember[mfsdata,key,targetset] extracts rows of an mfsdata object for which values labelled by key belong to targetset.

8.27 mfsMerge
mfsMerge[{qp1,qp2,...}] merges a list of mfs objects into a single one containing all the header and column information in each of them. The column lengths must match.

8.28 mfsNotMember
mfsNotMember[mfsdata,key,targetset] extracts rows of an mfsdata object for which values labelled by key DO NOT belong to targetset.

8.29 mfsRange
mfsRange[mfsdata,key,{min,max}] extracts rows of an mfsdata object for which values labelled by key lie within min and max. NOTE: min must be smaller than max !!!

8.30 mfsReverse
mfsReverse[mfsdata] returns the mfs object with the rows of the main block of columns in reverse order.

8.31 mfsSelect
mfsSelect[mfsdata,criterion] extracts rows satisfying criterion (function) from an mfsdata object.

8.32 mfsTypes

Message::name : Message name {mfs, SurvivalTable, TrackTable}::usage is not of the form symbol::name or symbol::name::language.

MessageName[{mfs, SurvivalTable, TrackTable}, usage]
8.33 mfsVersion2Update
mfsVersion2Update[oldfile,newfile] will create a new version of a file in which names of functions in Version 1.x
of the Madtomma'Mfs'Mfs' package are changed to their equivalents in Version 2.x.

8.34 mfsVersion2UpdateRules
   - Message::name :
     Message name \{interpretDescriptors \rightarrow \text{mfsInterpretKeys, \langle \langle 7 \rangle \rangle, symplecticJ \rightarrow \text{SymplecticJ}}\}::usage is not of the form symbol::name or symbol::name::language.

   MessageName[\{interpretDescriptors \rightarrow \text{mfsInterpretKeys, keyValue \rightarrow \text{mfsKeyValue, columnNames \rightarrow \text{mfsColumnNames, descriptorNames \rightarrow \text{mfsKeyNames, colValue \rightarrow \text{mfsColumnValue, addDescriptor \rightarrow \text{mfsAddKey, editDescriptor \rightarrow \text{mfsEditKey, deleteDescriptor \rightarrow \text{mfsDeleteKey, symplecticJ \rightarrow SymplecticJ}}}, usage]}

8.35 removeQuotes
removeQuotes[x] will remove any double quotes " explicitly included in a string x. If x is not a string then it is
returned unchanged.

8.36 removeUnwantedLines
removeUnwantedLines[infile,outfile,string] copies file infile to outfile, removing all lines containing string.

8.37 stringRemoveLeadingTrailingBlanks
stringRemoveLeadingTrailingBlanks[string] removes leading and trailing blanks from a string.

8.38 tfsFormatRules
tfsFormatRules is an option of tfsRead and related functions like tfsParseHeaderBlock. It gives a set of rules for
transforming TFS column formats into Mathematica data types.

8.39 tfsInterpretDescriptorLine
tfsInterpretDescriptorLine[string] takes a TFS descriptor line as a string, creates a variable from the key name and
assigns it the value.

8.40 tfsParseDescriptorLine
tfsParseDescriptorLine[string] takes a TFS descriptor line as a string and returns a list consisting of the TFS key
and its value.

8.41 tfsParseHeaderBlock
tfsParseHeaderBlock[file] returns the information in the header block of a TFS file as a structured list. It is
normally used inside readTfsTable.

8.42 tfsRead
tfsRead[file] returns an mfs data object containing all the information in a TFS file.

8.43Verbose
Verbose is an option for removeUnwantedLines, tfsParseHeaderBlock and tfsRead that specifies whether
informative messages should be printed.
9 User’s Guide and Examples

This section runs through the basic capabilities of the package using a small sample data file (based on very-short term tracking of a kicked beam in LEP). The operations performed are also kept rather simple applying one or two functions at a time. A realistic application may deal with many much larger data files. If required, the functions may be combined in arbitrarily complex ways, both with each other and with the many other functions available in the Mathematica environment.

9.1 Creation of a TrackTable object

The following expression finds the sample data file on any computer system (no need to understand it)

```mathematica
sampleFile = ToFileName[
    Flatten[{
        packageSourceDirectory["TrackTable"], "Examples"},
        "qkick.tfs"]
    P:\cern.ch\user\j\jowett\public\math\Madtoma\Mfs\Examples\qkick.tfs
]
```

Create and move to a directory of your own to use as working space for the examples, e.g.,

```mathematica
workDir = ToFileName[{$TemporaryPrefix, "TrackTableWorkspace"}]
```

```mathematica
CreateDirectory[workDir]
```

```mathematica
SetDirectory[workDir]
```

To create a TrackTable data object from a file, first create an mfs object with the function tfsRead [4], then apply makeTrackTable to it.

```mathematica
qpt = makeTrackTable[tfsRead[sampleFile, Verbose -> False]]; 
```

```mathematica
- mfsKeyValue::notfound : Descriptor T not found.
- mfsKeyValue::notfound : Descriptor PT not found.
```

The error messages are harmless and explained in "Warnings" above.

9.2 Extracting information

Show the list of data types in the Mfs class:
Now most of function defined in the Mfs.m package can be used for all mfsTypes data objects. This is also indicated in their usage messages. For example, the usage message for mfsColumn is now longer:

\texttt{mfsColumn[mfsdata,colname]} returns the column of data labelled by the string colname in an mfs (or related) data object. A list of colnames may also be given to return a set of columns. If colname is absent the entire block of columns is returned. \texttt{mfsColumn[TrackTableData,n,colname]} extracts a list of coordinates of particle n labelled by the string colname from a TrackTable data object. A list of colnames may also be given to return a set of lists. If colname is absent the entire block of coordinates is returned. If the particle number is absent a list of the requested column for all the particles is returned. \texttt{mfsColumn[SurvivalData,colname]}, \texttt{mfsColumn[SurvivalData, particle,colname]}, \texttt{mfsColumn[SurvivalData]} are the obvious extensions to SurvivalTable objects.

Find out what is in the object used for the examples.

\texttt{mfsKeyNames[qpt]}  
\texttt{\{TYPE, COMMENT, ORIGIN, DATE, TIME, TURNS, PARTICLES, EMITTANCES, ORBIT, EIGENVECTORS\}}

Note that this list is different from that of the intermediate mfs object because the closed orbit, eigenvector components and emittances have been combined into vectors and matrices.

\texttt{mfsKeyNames[tfsRead[sampleFile, Verbose \to False]]}  

There is information about the number of particles and the maximum number of turns survived by any of them.

\texttt{mfsKeyValue[qpt, \{"PARTICLES", "TURNS"\]}}  
\texttt{\{16, 109\}}
and the emittances that may have been used to define units for the initial conditions:

```plaintext
mfsKeyValue[qpt, "EMITTANCES"]
{2.98257 x 10^-8 Meter, 1.05428 x 10^-10 Meter, 0.0000102626 Meter}
```

Alternatively, as with `mfs` objects, we can create variables from the TFS header block (this may be convenient when we have only one `TrackTable` object to deal with).

```plaintext
mfsInterpretKeys[qpt];
```

These can then be used in calculations

```plaintext
EMITTANCES
{2.98257 x 10^-8 Meter, 1.05428 x 10^-10 Meter, 0.0000102626 Meter}
```

Show the names of the data columns in our example `TrackTable` object `qpt`.

```plaintext
mfsColumnNames[qpt]
{X, PX, Y, PY, T, DELTAP}
```

### 9.3 Closed orbit

This does not work now, although it would have worked with the `mfs` object from which the `TrackTable` object was derived.

```plaintext
mfsKeyValue[qpt, "{X,PX}"]

-- mfsKeyValue::notfound : Descriptor {X,PX} not found.
```

Instead, now we have to work with vectors.

```plaintext
mfsKeyValue[qpt, "ORBIT"]
{-0.000388354, -0.0000167029,
 -0.000381844, -2.94935 x 10^-6, Null, Null}
```

The MAD Version used to create the sample file, has a known inconsistency in that the last two components of the closed orbit are not transmitted to the TRACK table file. This should be fixed in later versions.

```plaintext
mfsKeyValue[qpt, "ORIGIN"]
MAD 8.21/11 RS6000 - AIX
```

### 9.4 Phase Space Coordinates

Extract data from qpt, for example plot the x-coordinate of the trajectory of particle number 3.
The `mfsColumn` function is listable in the sense that it can operate on lists of column names as its second argument. We can get the coordinates in synchrotron phase space from

```math
\text{tpt} = \text{mfsColumn}[\text{qpt}, 3, \{"T", "DELTAP"\}];
```

and plot them with the help of another package [5]

```math
\text{Needs["Madtomma\text{\text{\text{"}}Mfs\text{\text{\text{"}}\text{\text{\text{"}}}ColorListPlot\text{\text{\text{"}}}}"]}
```

```math
\text{colorListPlot[Transpose[\text{tpt}],}
\text{ Prolog -> \{PointSize[0.02]\}, \text{PlotRange -> All, PlotJoined -> True]}
```

- Graphics -
The particles were all started off on the closed orbit except for a radial kick. This picture shows the effect of Radiative Beta-synchrotron Coupling (RBSC) into the longitudinal plane (including quantum fluctuations).

### 9.5 Spectral analysis

The `powerSpectrum` function calculates the power spectrum Fourier transform of coordinates for a given particle coordinate and automatically truncates the upper (reflection) part of the spectrum. Here we use it for the $x$ coordinate of particle number 3.

```math
powerx = powerSpectrum[qpt, 3, "X"];
Short[powerx]
```

This abbreviated form of the output shows it is a list of ordered pairs of "tune" values and spectral densities.

```
ListPlot[powerx, PlotJoined -> True, PlotRange -> All]
```

The width of the peaks reflects the fact that the orbits in this example are short and strongly damped.

Like many other functions in the package, `powerSpectrum` also works, quite naturally, on simple lists.
The function `phaseSpectrum` does a similar job for the phases.

```math
phasex = phaseSpectrum[qpt, 3, "X"];
Short[phasex]

\{\{0, 0.\}, \{\frac{1}{109}, 1.70445\}, \ll52\rr, \{\frac{54}{109}, 6.53891\}\}
```

```math
ListPlot[phasex, PlotJoined -> True, PlotRange -> All]
```

- Graphics -
9.6 Eigenvectors and symplectic transformation matrix

The matrix formed from the eigenvectors gives the transformation from normalised variables to the usual canonical variables relative to the closed orbit. Here we print the result with reduced precision just for the sake of readability.

This matrix has unit determinant:

We can check symplecticity of the eigenvector matrix. See the documentation for the package Madtoma'Algebra'Symplectic' for further details.
SetOptions[SymplecticQ, SymplecticTolerance \to \frac{1}{10^{10}}]

\{SymplecticTolerance \to \frac{1}{10000000000}\}

SymplecticQ[eM]
True

Compute the inverse of the transformation:

MatrixForm[eMi = Chop[SymplecticInverse[eM]]]

\[
\begin{pmatrix}
0.190233 & 0 & -0.00652174 & -0.0846274 & 0.000189745 & 0.00505032 \\
-0.0115639 & 5.25333 & -0.00148939 & -0.118845 & 0.000102915 & -0.0174585 \\
0.004404 & -0.0829796 & 0.186545 & 0 & 0.000342355 & 0.018883 \\
-0.00199452 & 0.185018 & -0.00303821 & 5.35717 & 0.000117069 & -0.00377175 \\
0.00137726 & 0.0111704 & 0.000242386 & 0.00337974 & 0.420459 \\
0.000414979 & -0.00265101 & 0.000511784 & -0.00436134 & -0.00400956 & 2.37837
\end{pmatrix}
\]

Of course, this is the matrix used in the transformation from the original physical coordinates to the normalised coordinates. We can verify the inversion with

MatrixForm[Chop[eMi.eM]]

\[
\begin{pmatrix}
1. & 0 & 0 & 0 & 0 & 0 \\
0 & 1. & 0 & 0 & 0 & 0 \\
0 & 0 & 1. & 0 & 0 & 0 \\
0 & 0 & 0 & 1. & 0 & 0 \\
0 & 0 & 0 & 0 & 1. & 0 \\
0 & 0 & 0 & 0 & 0 & 1.
\end{pmatrix}
\]

9.7 Standard Canonical Transformations

The function toNormalCoords performs a transformation into normalized coordinates (projections on the eigenvectors), returning a new TrackTable object. Note that, in this representation, all coordinates and momenta are in units of \sqrt{\text{Meter}}.

\[qptN = toNormalCoords[qpt];\]

- General::spell1 : Possible spelling error:
  new symbol name "qptN" is similar to existing symbol "qpt".

The column names have changed suitably.

mfsColumnNames[qptN]
\{'XN, PXN, YN, PYN, TN, PTN\}
Now one can investigate particle trajectories in transformed coordinates.

```
ListPlot[mfsColumn[qptN, 3, "TN"], PlotJoined -> True]
```

A similar function transforms the data into the action-angle variables of the linearised motion:

```
qptAA = toActionAngle[qptN];
```

Again, the column names are changed.

```
mfsColumnNames[qptAA]
{IX, PHIX, IY, PHIY, IT, PHIT}
```

With RBSC, the synchrotron phase behaves quite unlike that of a harmonic oscillator.
The action variable for horizontal betatron motion decays, more-or-less exponentially.

It is sometimes helpful to combine coordinates from two different representations. Here we plot the first action variable against the synchrotron coordinate $T$, using colour to see how the motion evolves.
9.8 Lost particles and beam intensity

Typically, tracked particles are considered independently of each other (e.g. when finding a dynamic aperture). At other times, it is useful to consider the ensemble of tracked particles in a TRACK table as a whole, representing the evolution of a "beam". Here we provide some functions relating to the intensity and particle losses that may be useful in such applications.

The examples up to here relied on particle 3 because it was the only one to survive to the end of the tracking. When a particle is lost, its coordinates are replaced by Null
This can be inconvenient, e.g., when plotting, so the function \texttt{notLost} is provided to extract the orbit up to the turn where the particle is lost:

\begin{verbatim}
notLost[mfsColumn[qptAA, 15, "IX"]]
{3.44968 \times 10^{-6}, 2.9683 \times 10^{-6}, 4.87348 \times 10^{-6}, 3.36005 \times 10^{-6},
2.74196 \times 10^{-6}, 3.23606 \times 10^{-6}, 2.34496 \times 10^{-6}, 2.30612 \times 10^{-6},
2.61267 \times 10^{-6}, 2.63516 \times 10^{-6}, 3.06301 \times 10^{-6}, 2.26977 \times 10^{-6},
2.25507 \times 10^{-6}, 2.46357 \times 10^{-6}, 2.62588 \times 10^{-6}, 2.59908 \times 10^{-6},
1.95829 \times 10^{-6}, 1.81604 \times 10^{-6}, 1.97793 \times 10^{-6}, 2.57183 \times 10^{-6},
2.79727 \times 10^{-6}, 2.27533 \times 10^{-6}, 2.05669 \times 10^{-6}, 2.27412 \times 10^{-6},
2.31356 \times 10^{-6}, 2.44856 \times 10^{-6}, 2.76803 \times 10^{-6}, 2.01099 \times 10^{-6},
1.99461 \times 10^{-6}, 1.92046 \times 10^{-6}, 8.78633 \times 10^{-7}, 2.42862 \times 10^{-6}}
\end{verbatim}

Another function returns the number of turns particle 4 did before being lost:

\begin{verbatim}
turnsSurvived[qpt, 4]
19
\end{verbatim}

We can also get the fraction of the original number of particles surviving at each turn. The function \texttt{intensity} gives the fraction of particles surviving on each turn

\begin{verbatim}
Short[decay = intensity[qpt], 6]
{1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1}]
\end{verbatim}

and makes it easy to plot the "decay of the beam"
9.9 Survival data

For many purposes it is not necessary to use all the data on turn-by-turn particle coordinates in a \texttt{TrackTable} object. It may be enough to know how many turns particles survived for. This is the role of the corresponding \texttt{SurvivalTable} object created by the function \texttt{toSurvival}:

\begin{verbatim}
qps = toSurvival[qpt];
\end{verbatim}

It contains the same header information but only the initial conditions and the number of turns survived. These are extracted in similar ways. In our example all particles have identical initial conditions

\begin{verbatim}
\texttt{mfsColumn[\texttt{qps, "PX"}]}
(0.0005, 0.0005, 0.0005, 0.0005, 0.0005, 0.0005, 0.0005, 0.0005, 0.0005, 0.0005, 0.0005, 0.0005, 0.0005, 0.0005, 0.0005, 0.0005)
\end{verbatim}

A new column name has been added for the number of turns survived.
mfsColumn[qps, "SURVIVED"]
{17, 11, 109, 19, 11, 12, 40, 13, 15, 12, 11, 25, 13, 16, 32, 24}

In deterministic tracking this can be used to produce many kinds of "survival plots". Here, with the help of some standard packages, we can make a plot of the "lifetime distribution"

Needs["Graphics`Graphics`"]

Needs["Statistics`DataManipulation`"]

midpoints = Table[k + 5, {k, 0, 100, 10}];

BarChart[Transpose[
{BinCounts[mfsColumn[qps, "SURVIVED"], {0, 110, 10}], midpoints}]]
It is often useful to combine survival data from many tracking jobs to compile a picture of the dynamic aperture.

### 9.10 Strobing

```plaintext
? strobe
```

```plaintext
strobe[list, {int1, int2}] samples list giving every int1-th element starting from the element number int2.
```

Take a list of momenta of particle number 4:

```plaintext
px = notLost[mfsColumn[qpt, 4, "PX"]]
{0.0005, -0.000417859, 0.000489193, -0.000189486, 0.0000353001, 0.000188357, -0.000382664, 0.000397831, -0.000038653, 0.000355385, -0.000314478, 0.000259996, -0.0000465206, -0.0000275869, 0.000390829, -0.000400594, 0.000332015, -0.000216295, 0.000278229}
```

We can extract extract every 3rd element of it starting from element number 1.

```plaintext
strobe[px, {3, 1}]
{0.0005, -0.000189486, -0.000382664, 0.000355385, -0.0000465206, -0.0000400594}
```

This can be useful when studying resonant phenomena.

### 9.11 Centre of gravity motion

To compare with experiment it is useful to calculate the centre-of-gravity motion of an ensemble of particles. If some of them are lost in the process, they will carry zero weight. Thus it is appropriate to substitute zeros for the Nulls corresponding to lost particles. These operations are included in the function `centreOfGravity`.

Calculate the y coordinate of the centre-of-gravity motion.
cgpy = centreOfGravity[qpt, "Y"];  

colorListPlot[cgpy, PlotRange -> All, Prolog -> {PointSize[0.02]}]

You can also call this function with a list of column names, eg. x and px.

cgxpx = centreOfGravity[qpt, {"X", "PX"}];

colorListPlot[Transpose[cgxpx],  
Prolog -> {PointSize[0.02]}, PlotRange -> All]

Now calculate the centre-of-gravity motion for the action variable (see Examples7.6).

cgIX = centreOfGravity[qptAA, "IX"];
Using the standard function \texttt{Fit}, you can try to fit the data to an exponential decay and extract a damping time:

\[
\text{th} = \text{Fit}[\log[\text{cgIX}], \{1, \text{turn}\}, \text{turn}]
\]
\[
12.9834 - 0.0484749 \text{turn}
\]

\[
\text{dampingTime} = \frac{1}{\text{Coefficient[th, turn]}}
\]
\[
20.6292
\]

\[
\text{plot2 = Plot}[e^{\text{th}}, \{\text{turn, 1, 109}\}, \text{DisplayFunction} \to \text{Identity}];
\]
\[
\text{Show[plot1, plot2]}
\]
Clearly, the fit to an exponential decay was not really justified in this case because of resonant behaviour.

9.12 Saving TrackTable objects

We can save the data in the new form as an alternative to the original TFS file. This will load faster but takes up some more disk space.

\[
\text{Save["qpt.m", qpt]}
\]

9.13 Avoiding typing and remembering commands

Mathematica’s palettes provide a convenient way to input commands without having to remember their names or syntax. They also help considerably to reduce syntax errors when you build up complex expressions. By selecting it ant then using the \texttt{File/Generate Palette from Selection} command, the following input cell can be transformed into a palette containing templates for the most useful commands in this package.

```
tfsRead[]
makeTrackTable[]
mfsColumn[[], {}, {}]
mfsColumn[[], {}, {}, {}]
notLost[]
sJ3
toSurvival[]
toNormalCoords[]
toActionAngle[]
strobe[[], {, }]
intensity[]
turnsSurvived[],
centreOfGravity[[], ]
powerSpectrum[[], , ]
```

The option \texttt{ShowGroupOpenCloseIcon->True} is useful in palettes.

Note that the palette buttons are most useful if you first select a part of an expression on your screen to which you want to apply them. For example (in Windows, place the cursor anywhere in the word \texttt{mfsColumn} and press Ctrl-. twice) selecting the whole expression \texttt{mfsColumn[toNormalCoords[qpt],3,\texttt{"XN"}]} in
colorListPlot[notLost[mfsColumn[toNormalCoords[qpt], 3, "XN"]]]

![Graph](image)

- Graphics -

And then clicking the `strobe[●, {□, □}]` button, will wrap the function around it (at the position of the selection placeholder ●) to give

```
colorListPlot[
    notLost[strobe[mfsColumn[toNormalCoords[qpt], 3, "XN"], {□, □}]]
```

with the selection already moved to the next argument slot □. The last two arguments can then be filled in (moving between them with the TAB key)

```
colorListPlot[
    notLost[strobe[mfsColumn[toNormalCoords[qpt], 3, "XN"], {3, 1}]]
```

![Graph](image)

- Graphics -
10 Bibliography


   http://www.wolfram.com/~maeder/ProgInMath/

   http://wwwslap.cern.ch/~fci/mad/mad.html

   http://wwwslap.cern.ch/~jowett/Madtomma/Mfs/Mfs.html

   http://wwwslap.cern.ch/~jowett/Madtomma/Mfs/ColorListPlot.html