Prototype Impedance Database

John Jowett, ABP-RLC meeting 21/11/2006

At the last RLC meeting, I suggested that *Mathematica* might be a good medium for a new implementation of an impedance database.

It was clear that not everyone is familiar with the idea of implementing databases in *Mathematica* nor what the advantages might be.

I spent a short time looking at the existing data and looking at the basics of an implementation which I will demonstrate today.

This is NOT a finished product, just a "proof-of-principle".

Nevertheless, I do not think it would be a great deal of work to complete the job. It could be a project for a Technical Student or even a very good Summer Student.

Of course we still need to enforce social and managerial aspects.

Database

From Wikipedia, the free encyclopedia

For other senses of this word see database (disambiguation).

The term **database** originated within the computer industry. Although its mea
even to include non-electronic databases, this article takes a more technical
database is a collection of records stored in a computer in a systematic way
it to answer questions. The items retrieved in answer to queries become infor
decisions. The computer program used to manage and query a database is k
system (DBMS). The properties and design of database systems are include

The central concept of a database is that of a collection of records, or pieces
database, there is a structural description of the type of facts held in that dat
schema. The schema describes the objects that are represented in the datal
There are a number of different ways of organizing a schema, that is, of mode
known as database models (or data models). The model in most common us
layman's terms represents all information in the form of multiple related table
(the true definition uses mathematical terminology). This model represents re
to more than one table. Other models such as the hierarchical model and the
representation of relationships.

For an impedance database, I believe it is more important to think about the *Schema* than the relations or the tables.

Requirements

■ Primary (physics-driven)

Provide up-to-date impedance and/or wake *functions* for all elements in the LHC.

Integrity of data related to LHC (layout, optics, occurrences of elements,...).

Many kinds of calculations combining impedances of elements, physical formulas for growth rates, etc.

■ Secondary (computer-driven)

Contain diverse sorts of data ("formats", unstructured output from various programs like ABCI, URMEL, et al).

Access to data from all platforms, across Internet, from control system, …

Maintain means for *updating* impedances (input files for programs, book-keeping, convenient interface).

As simple as possible, future-proof. Minimise dependences on other software.

Use existing data from previous implementation (or change it once and for all to make life a little easier?).

Proposed Model

Impedance data is stored in a file system, readable via the Web. Can be organized in a hierarchy

```
{"machine", "element group", "item", "mode"}
```

which could be implemented as a hierarchy of files as at present.

There can be two kinds of users

■ Database maintainer (modifies data)

Responsible for updating calculations or inserting results provided by others.

Access direct (file system)

Interface is typically a *Mathematica* notebook.

■ Application (read-only)

Responsible for updating calculations or inserting results provided by others.

Access can be via Web.

Interface is typically a *Mathematica* notebook for a particular application which may contain other kinds of material (impedances may be just one ingredient of what the user is doing).

Unimportant Technicalities

This demonstration is on a laptop without network access so I am using (partial) local copies of the existing Zbase data files and the LHC optics database.

The existing implementation contains a number of compressed (.gz) files with inconsistent file extensions. I have uncompressed and renamed some of them for this demonstration.

I will not show how to submit batch jobs for impedance calculations. But this is straightforward from a *Mathematica* kernel running on, e.g., the lxplus system.

Note that this is a *development* notebook which contains rather more "code" (trial function definitions etc.) than an eventual *application* notebook would normally contain. So there is no need to be put off by this. In *Mathematica*, correct code is entered with many fewer keystrokes than appear necessary anyway.

A more "graphical" user interface (with buttons, etc.) can also be created. However full flexibility will be retained with access to the package functions.

Setup

If this is an application notebook, the user would normally just load the package with a single command.

At present the implementation of the functions is in another notebook which has already been evaluated. As usual, every function should have a usage message (instant documentation). So far just a few functions have been implemented.

In[135]:=
```
?$Zbase*
```

Global`

```
$ZbaseDirectory  $ZbaseModes
```

In[136]:=
```
?Zbase*
```

Global`

```
Zbase              ZbaseFiles  ZbaseItems    ZbaseMachines
ZbaseDataDirectory ZbaseItem   ZbaseMachine  ZbaseMode
```

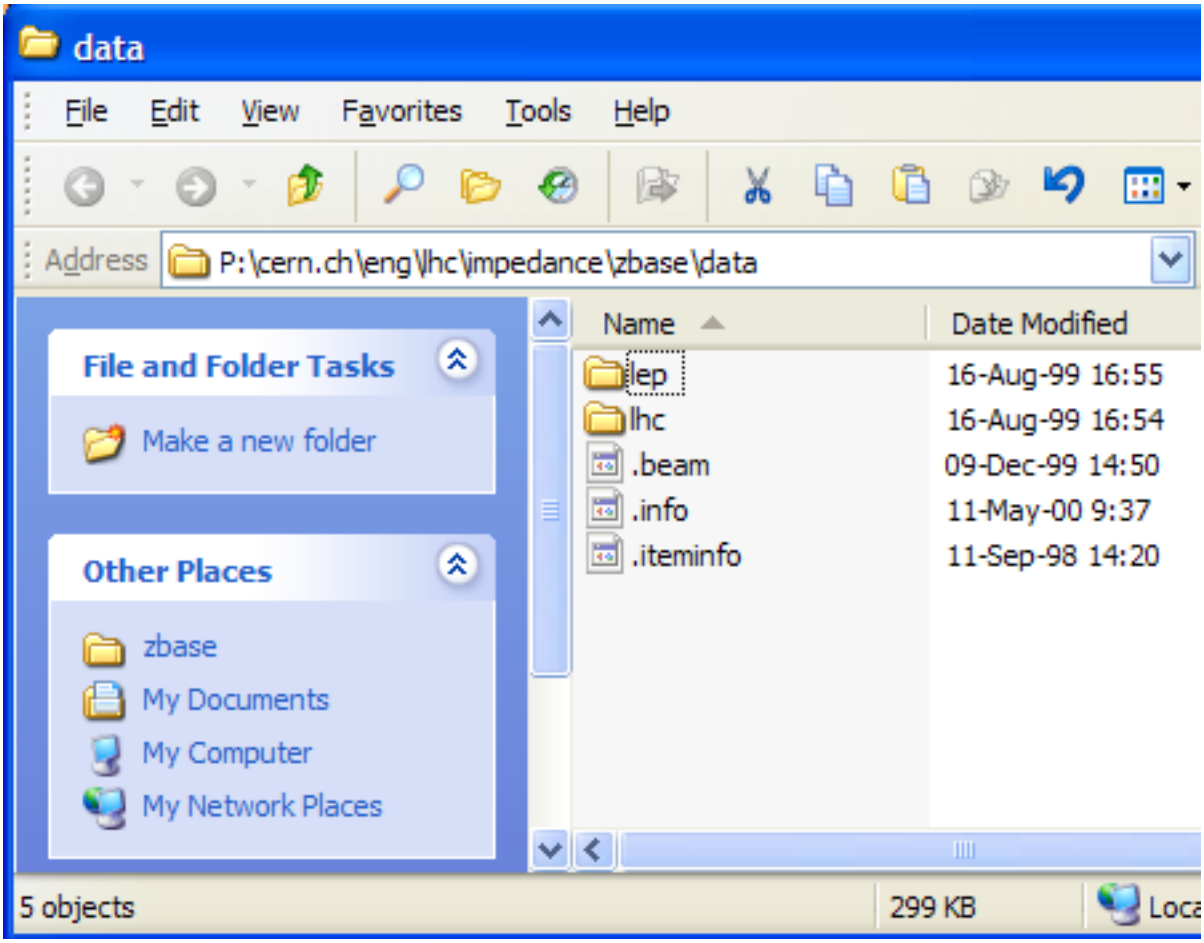
```
ZbaseMachines[] returns a list of all machines stored in the Zbase database.
```

In the talk I used a local copy of database. This is the real one:

In[137]:=
```
ZbaseDataDirectory[] // explorer
```

Out[137]=
```
P:\cern.ch\eng\lhc\impedance\zbase\data\
```

causes this to appear on screen



Database Contents

In[138]:=

```
Options[Zbase]
```

Out[138]=

```
{ZbaseMachine → lhc, ZbaseItem → {cavities, scavity}, ZbaseMode → long}
```

In[139]:=

```
ZbaseMachines[]
```

Out[139]=

```
{lep, lhc}
```

In[140]:=

```
ZbaseItems[]
```

Out[140]=

```
{{bellows-us, collimator}, {bpm, electrode}, {cavities, scavity},
 {experiments, alice}, {experiments, alice-nozdc}, {experiments, alice-vers1-jul98},
 {experiments, alice-vers2-jul98}, {experiments, alice-vers3-jul98},
 {experiments, alice-vers4-dec99}, {experiments, alice-zdc},
 {experiments, atlas}, {experiments, atlas-urmel}, {experiments, cms},
 {experiments, cms-dec97}, {experiments, cms-dec97-0.4tap},
 {experiments, cms-dec97-1.4tap}, {experiments, cms-dec97-20cm},
 {experiments, cms-dec97-20cm-10cmtap}, {experiments, cms-dec97-20cm-20cmtap},
 {experiments, cms-highlumi2-april2000}, {experiments, cms-highlumi-april2000},
 {experiments, cmshighlumi-mafia}, {experiments, cmshighlumi-stst-mafia},
 {experiments, cmshighlumi-urmel}, {experiments, cmslowlumi-mafia},
 {experiments, cmslowlumi-stst-mafia}, {experiments, cmslowlumi-urmel},
 {experiments, cms-totem}, {experiments, cms-totem-20cmtaper},
 {experiments, cms-totem-50cmtaper}, {experiments, cms-withrfscreen},
 {experiments, felix2}, {experiments, felix2-11cmscr}, {experiments, felix2-6cmscr},
 {experiments, felix2-8cmscr}, {experiments, felix2-8cmtap},
 {experiments, felix2-9cmscr}, {experiments, felix-centralpart},
 {experiments, felix-outerpart}, {experiments, lhc-b}}
```

In[141]:=

```
ZbaseItems[{"experiments"}]
```

Out[141]=

```
{{alice}, {alice-nozdc}, {alice-vers1-jul98}, {alice-vers2-jul98},
 {alice-vers3-jul98}, {alice-vers4-dec99}, {alice-zdc}, {atlas}, {atlas-urmel},
 {cms}, {cms-dec97}, {cms-dec97-0.4tap}, {cms-dec97-1.4tap}, {cms-dec97-20cm},
 {cms-dec97-20cm-10cmtap}, {cms-dec97-20cm-20cmtap}, {cms-highlumi2-april2000},
 {cms-highlumi-april2000}, {cmshighlumi-mafia}, {cmshighlumi-stst-mafia},
 {cmshighlumi-urmel}, {cmslowlumi-mafia}, {cmslowlumi-stst-mafia},
 {cmslowlumi-urmel}, {cms-totem}, {cms-totem-20cmtaper}, {cms-totem-50cmtaper},
 {cms-withrfscreen}, {felix2}, {felix2-11cmscr}, {felix2-6cmscr}, {felix2-8cmscr},
 {felix2-8cmtap}, {felix2-9cmscr}, {felix-centralpart}, {felix-outerpart}, {lhc-b}}
```

In[142]:=

```
ZbaseFiles[{"bellows-us", "collimator"}, "long"] // TableForm
```

Out[142]//TableForm=

```
abci.lepbellow                  File
abci.lepbellow.ps.gz            File
.loss                           File
.wakepot.abci.long0.7.5e-2.gz   File
.wakepot.abci.longx.7.5e-2.gz   File
.zbase                          File
```

Contents of the .zbase file

- First, there is one file which contains some basic information on the impedance item. In the following and in the **ZBASE** program we will call these entries the **item attributes**. The information is stored in a file named **.zbase**. The first entry is a comment line describing the item. The second entry is the number of impedance components that are in the machine. The third and fourth entries give the average horizontal and vertical betatron values in meters at the location of the impedance component in the machine. The fifth entry is the low frequency inductance (Z_0/n) in Ω and the sixth and seventh entries give the horizontal and vertical low frequency limit of the imaginary part of the transverse impedance ($Z_{x,\perp}$ and $Z_{y,\perp}$) in Ω/m. The eighth entry specifies the location of the item in the machine (→ Section 7.5). Whenever a new item is entered into the data base, the program will create the **.zbase** file and ask the user to supply the required information.
- The second type of data are the higher-order-mode entries (e.q: 'frequency', 'R/Q' and 'Q-value') which are also stored in the **.zbase** file (→ Section 4). The mode frequencies must be specified in Hz and the R/Q values in Ω. The program expects all entries in standard **ANSI-C** format.

Pick an example of this kind of file

```
In[143]:=
zbaseFile =
 ToFileName[{ZbaseDataDirectory[], "lhc", "cavities", "scavity", "long"}, ".zbase"]
```

```
Out[143]=
P:\cern.ch\eng\lhc\impedance\zbase\data\lhc\cavities\scavity\long\.zbase
```

Importing the file as a Table is an easy way to deal with the E-format numbers.

```
In[63]:= zbaseFileContents = Import[zbaseFile, "Table"];
```

```
In[64]:= TableForm[zbaseFileContents]
```

Example of how we might represent this information in *Mathematica*:

```
In[144]:=
zbaseFileRules = {"COMMENT" → StringJoin @@ ToString[zbaseFileContents[[1]]],
   NumberOfItems → zbaseFileContents[[2]],
   betxav → zbaseFileContents[[3]],
   betyav → zbaseFileContents[[4]],
   Zovern → zbaseFileContents[[5]],
   HOMs → Drop[zbaseFileContents, 10]}
```

```
Out[144]=
{COMMENT → {8, Cells, of, the, Superconducting, accleleration, Cavities},
 NumberOfItems → {1}, betxav → {80.}, betyav → {80.}, Zovern → {0.00034},
 HOMs → {{3.96769×10^8, 25.021, 43898}, {3.96769×10^8, 64.7479, 43898},
   {3.97206×10^8, 4.47258, 44160}, {3.97206×10^8, 85.2469, 44160},
   {3.9731×10^8, 13.2046, 43616}, {3.9731×10^8, 76.012, 43616},
   {3.97312×10^8, 1.32437, 43616}, {3.97312×10^8, 88.0589, 43616},
   {7.63797×10^8, 0.0606469, 57626}, {7.6428×10^8, 0.0232749, 57821},
   {7.6428×10^8, 0.302223, 57821}, {7.66707×10^8, 0.018773, 58238},
   {7.67448×10^8, 0.10212, 59302}, {7.67463×10^8, 0.975679, 59364},
   {7.68279×10^8, 0.290241, 59661}, {7.68279×10^8, 0.339455, 59663},
```

```
{7.6891×10^8, 0.000162596, 59433}, {7.68912×10^8, 0.0166369, 59449},
{7.71398×10^8, 0.546113, 59276}, {7.75451×10^8, 2.87035, 61231},
{7.75451×10^8, 0.0460074, 61231}, {7.75492×10^8, 0.809371, 60296},
{7.75492×10^8, 1.01336, 60297}, {7.77702×10^8, 0.120843, 60224},
{7.81124×10^8, 2.89539, 62855}, {7.81138×10^8, 0.945862, 62634},
{7.82617×10^8, 1.32124, 63987}, {7.82617×10^8, 1.50073, 63986},
{7.85147×10^8, 1.38204, 61539}, {7.85153×10^8, 0.0181562, 61525},
{7.85772×10^8, 0.583636, 60450}, {7.95768×10^8, 0.999874, 60780},
{7.9756×10^8, 0.347863, 61267}, {7.97561×10^8, 0.0202092, 61271},
{8.06342×10^8, 0.139331, 63488}, {8.06913×10^8, 0.0129476, 62203},
{8.08212×10^8, 0.0224959, 63149}, {8.08212×10^8, 0.166502, 63159},
{8.08657×10^8, 0.0312785, 59357}, {8.13926×10^8, 0.627648, 60573},
{8.13926×10^8, 2.30886, 60573}, {8.15503×10^8, 0.209193, 60248},
{8.15503×10^8, 1.72296, 60248}, {8.22659×10^8, 0.792564, 60311},
{8.33773×10^8, 0.872853, 60828}, {8.33773×10^8, 0.119769, 60827},
{8.39257×10^8, 0.625017, 60793}, {8.52535×10^8, 2.61033, 61438},
{8.52749×10^8, 2.85894, 60662}, {8.53438×10^8, 5.64209, 61322},
{8.53438×10^8, 0.257206, 61311}, {8.57389×10^8, 0.120681, 60361},
{8.5739×10^8, 4.5746×10^-6, 60337}, {8.58184×10^8, 0.3765, 60238},
{8.78512×10^8, 0.317605, 61328}, {8.83717×10^8, 1.15286, 61564},
{8.83717×10^8, 1.03366, 61564}, {8.94645×10^8, 1.36401, 59108},
{8.94645×10^8, 0.548724, 59108}, {9.00569×10^8, 1.06849, 62048},
{9.12741×10^8, 0.123358, 62210}, {9.12741×10^8, 3.16276, 62210},
{9.17573×10^8, 2.43465, 61879}, {9.17573×10^8, 2.95631, 61855},
{9.17777×10^8, 1.0057, 61606}, {9.17777×10^8, 4.80756, 61606},
{9.2402×10^8, 0.812994, 62761}, {9.43925×10^8, 0.0860959, 63449},
{9.43925×10^8, 0.721883, 63450}, {9.48633×10^8, 0.0800181, 63528},
{9.7411×10^8, 0.284496, 64047}, {9.76473×10^8, 0.113747, 63934},
{9.76481×10^8, 0.0446832, 63980}, {9.80123×10^8, 1.22598, 58328},
{9.80123×10^8, 0.040943, 58328}, {9.93098×10^8, 2.26692, 63180},
{9.93578×10^8, 1.01494, 64190}, {9.93983×10^8, 0.0298923, 65036},
{9.94152×10^8, 0.108374, 65356}, {1.00124×10^9, 0.281301, 66617},
{1.0111×10^9, 0.426292, 66551}, {1.01112×10^9, 0.455323, 66624},
{1.02755×10^9, 1.12507, 66431}, {1.04462×10^9, 0.54217, 66243},
{1.04468×10^9, 1.88363, 66366}, {1.05381×10^9, 0.94707, 66338},
{1.05664×10^9, 0.436474, 60557}, {1.05665×10^9, 1.83904, 60619},
{1.06886×10^9, 0.882464, 65651}, {1.0708×10^9, 2.15261, 66645},
{1.0741×10^9, 0.706266, 67513}, {1.07514×10^9, 0.327648, 68827},
{1.08263×10^9, 0.231686, 69478}, {1.08488×10^9, 1.24708, 71237},
{1.08736×10^9, 0.466344, 72183}, {1.10943×10^9, 0.288535, 68740},
{1.11449×10^9, 1.90546, 68959}, {1.11554×10^9, 0.307764, 70047},
{1.13032×10^9, 1.55592, 64459}, {1.13116×10^9, 0.0343205, 63569},
{1.13371×10^9, 0.0149164, 67386}, {1.14145×10^9, 0.15305, 68164},
{1.14515×10^9, 6.9452×10^-6, 68840}, {1.15102×10^9, 0.0192246, 69070},
{1.15482×10^9, 1.01382, 70741}, {1.16129×10^9, 1.90778, 71412},
{1.16433×10^9, 4.00241, 73682}, {1.17202×10^9, 2.87326, 73995},
{1.18375×10^9, 4.19917, 70145}, {1.18669×10^9, 1.08528, 71749},
{1.1941×10^9, 0.425108, 71570}, {1.20125×10^9, 0.0622168, 68200},
{1.20341×10^9, 1.91841, 70216}, {1.21041×10^9, 0.705541, 71474},
{1.21566×10^9, 1.06543, 72399}, {1.22055×10^9, 0.753186, 72816},
{1.22466×10^9, 0.202745, 73961}, {1.22826×10^9, 0.114645, 75941},
{1.23369×10^9, 0.461462, 75800}, {1.23691×10^9, 0.0405645, 76969},
{1.23664×10^9, 0.0358254, 76836}, {1.24203×10^9, 0.00385793, 78171},
{1.24074×10^9, 0.12275, 76398}, {1.25249×10^9, 0.498197, 76647},
{1.24756×10^9, 0.0897901, 74930}, {1.26425×10^9, 0.0945683, 72153},
{1.25931×10^9, 0.266855, 74106}, {1.28772×10^9, 0.073711, 74371}}}
```

These operations could be wrapped up as standard functions.

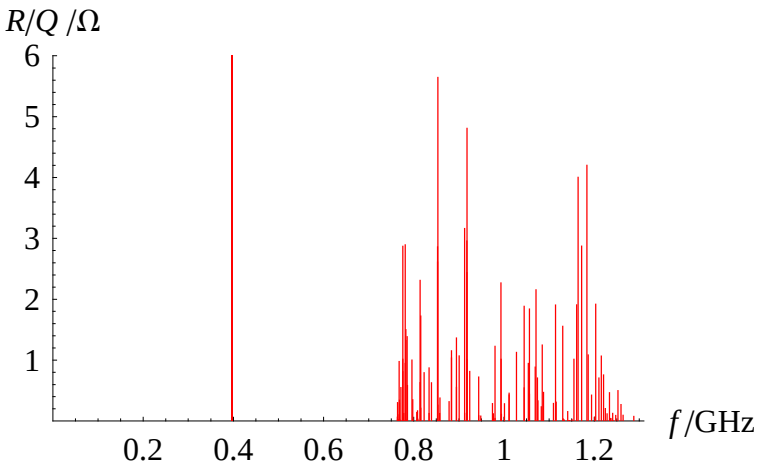
More on the .zbase file contents

Show the first few HOMs

```
In[145]:=
Take[HOMs /. zbaseFileRules, 5]

Out[145]=
{{3.96769×10^8, 25.021, 43898},
 {3.96769×10^8, 64.7479, 43898}, {3.97206×10^8, 4.47258, 44160},
 {3.97206×10^8, 85.2469, 44160}, {3.9731×10^8, 13.2046, 43616}}
```

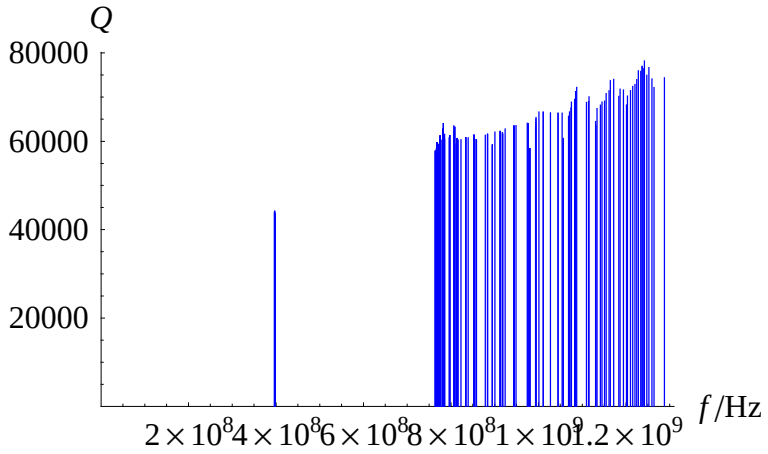
Plots, to be wrapped up as functions with proper option inheritance etc. Remove

```
In[150]:=
SpectralListPlot[HOMs /. zbaseFileRules /. {f_, rq_, _} → {f / 10^9, rq},
 PlotRange → {{0, Automatic}, {0, 6}}, PlotStyle → Red, AxesLabel → {"f/GHz", "R/Q /Ω"}]
```




```
Out[150]=
- Graphics -
```

```
In[151]:=
SpectralListPlot[HOMs /. zbaseFileRules /. {f_, rq_, q_} → {f, q},
 PlotRange → {{0, Automatic}, {0, Automatic}},
 PlotStyle → Blue, AxesLabel → {"f/Hz", "Q"}]
```




```
Out[151]=
- Graphics -
```

It will be straightforward to put together functions that display this kind of information, starting from the ZbaseItem and ZbaseMode values.

Contents of the .beam file

```
In[154]:=
exampleBeamFile = ToFileName[{ZbaseDataDirectory[], "lhc"}, ".beam"]

Out[154]=
P:\cern.ch\eng\lhc\impedance\zbase\data\lhc\.beam

In[155]:=
beam = Import[exampleBeamFile, "Table"]

Out[155]=
{{lhc}, {protons}, {2835}, {1.×10^11}, {0.13}, {0.075}, {479.6}, {7460.6},
 {0.001}, {0.000345}, {11245.8}, {62.}, {21.}, {2.495×10^-8}, {1.×10^10}}
```

We can identify the first 13 of these parameters by comparison with

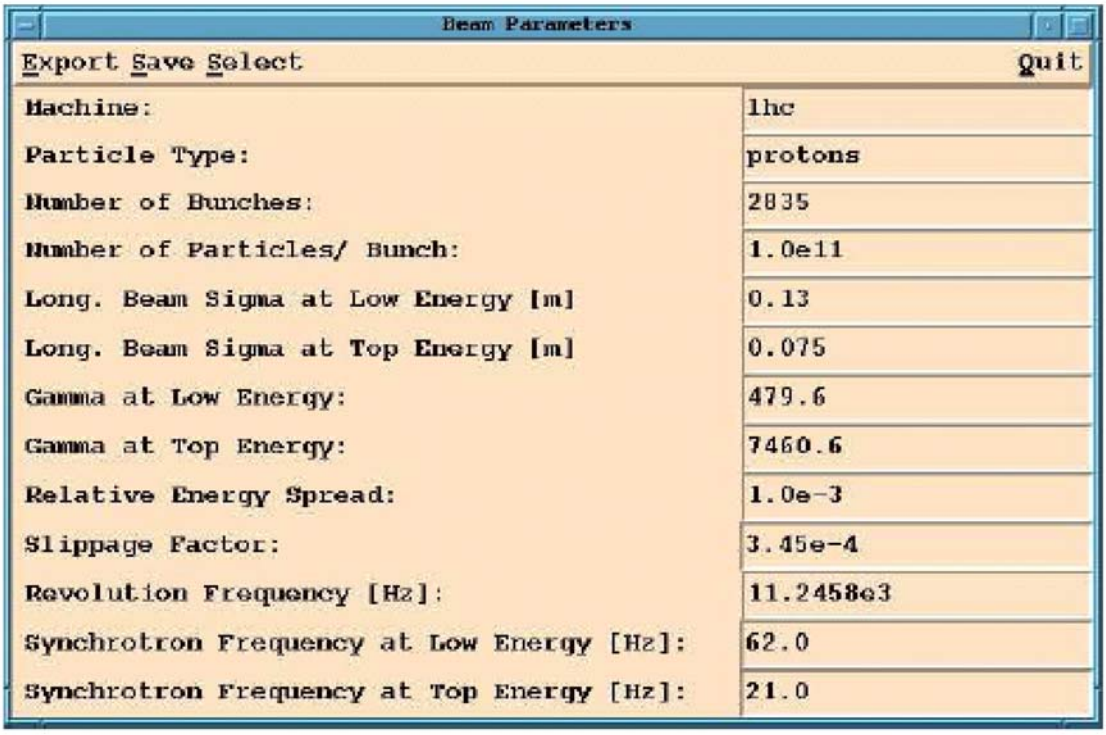


Figure 6: *Listing of the default parameters for one machine in the* **ZBASE** *data base.*

In *Mathematica*, we could store the information as a list of rules. This has the advantage of being self-describing. Something like this:

```
In[156]:=
     Rule @@@ Transpose@{{Machine, Particle, Kbunch, Nbunch,
         sigzInjection, sigzCollision, gammaInjection, gammaCollision, sige,
         slipFactor, frev, QsInjection, QsCollision, xx1, xx2}, Flatten[beam]}

Out[156]=
     {Machine → lhc, Particle → protons, Kbunch → 2835, Nbunch → 1. × 10^11,
      sigzInjection → 0.13, sigzCollision → 0.075, gammaInjection → 479.6,
      gammaCollision → 7460.6, sige → 0.001, slipFactor → 0.000345, frev → 11245.8,
      QsInjection → 62., QsCollision → 21., xx1 → 2.495 × 10^-8, xx2 → 1. × 10^10}
```


ABCI, URMEL, etc. files

I haven't yet implemented anything.

Probably store files used for a given impedance calculation as a list of rules corresponding to the element name.

Try to achieve one-to-many mapping of impedance element types to elements in machine layout (MAD description?).

Plugging in of specific stored parameters in template input files and launching of jobs. Can facilitate process and keep track of things.

Other sources of impedance information

Besides the output from programs like GDFIDL, ABCI, URMEL etc., we may have impedances given by analytical formulas. Make a scheme for including these - big advantage of *Mathematica*.

Generic formulas for typical impedances with rules giving specific values of parameters.

Probably good to use ConstantsUnits package to track units and dimensions of symbols appearing in the formulas.

A typical wake

Some of the data files are stored in gzipped archives (the .gz files). This is not very convenient and somewhat system-dependent. I think it would be better to change it.

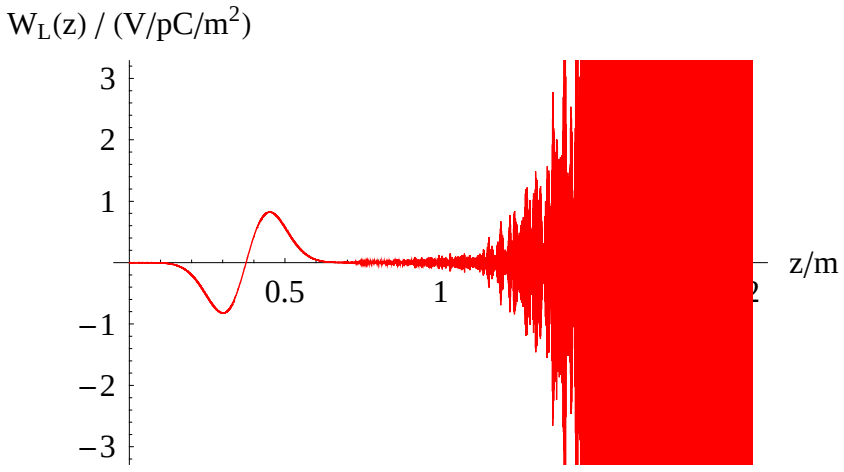
Meanwhile, for demonstration purposes, I extracted a data file containing a wake by hand (and renamed it)

```
In[75]:= sampleWakeFile = First[FileNames["*.dat"]]

Out[75]= wakepot.abci.longx.7.5e-2.dat

In[76]:= sampleWakeFileContents = Import[First[FileNames["*.dat"]]]

  Out[76]//Short=
       {{TITLE:, LONGITUDINAL, WAKE, POTENTIAL}, {LARGE, UNSHIELDED, LEP, BELLOWS, "VBDA"},
        {DATE:, TIME:, MROT:, NO., OF, POINTS:}, {01/11/96, 18.37.44, 1, 10001},
        {S, (m), WZ, (V/pC/m**2)}, {0., -0.00013341}, {0.0002, -0.00012249},
        {0.0004, -0.00014038}, {0.0006, -0.00017591}, {0.0008, -0.00020639},
        {0.001, -0.00022198}, {0.0012, -0.0002377}, {0.0014, -0.00024907},
        {0.0016, -0.00024221}, {0.0018, -0.00023269}, {0.002, -0.00021038}, <<9974>>,
        {1.997, -6985.4}, {1.9972, 1778.1}, {1.9974, 8353.8}, {1.9976, -3844.4},
        {1.9978, -8586.6}, {1.998, 5903.4}, {1.9982, 7977.7}, {1.9984, -7440.6},
        {1.9986, -7147.7}, {1.9988, 8591.1}, {1.999, 6392.8}, {1.9992, -9687.2},
        {1.9994, -5570.6}, {1.9996, 10744.}, {1.9998, 4556.4}, {2., -11526.}}
```

The first 5 lines of the files are not part of the wake so drop them before plotting.

```
In[88]:= sampleWakeData = Drop[sampleWakeFileContents, 5];
```

```
In[89]:= rawPlot = ListPlot[sampleWakeData, PlotStyle → Red,
           PlotJoined → True, AxesLabel → {"z/m", "W_L(z) / (V/pC/m^2)"}]
```




```
Out[89]//Short=
      - Graphics -
```

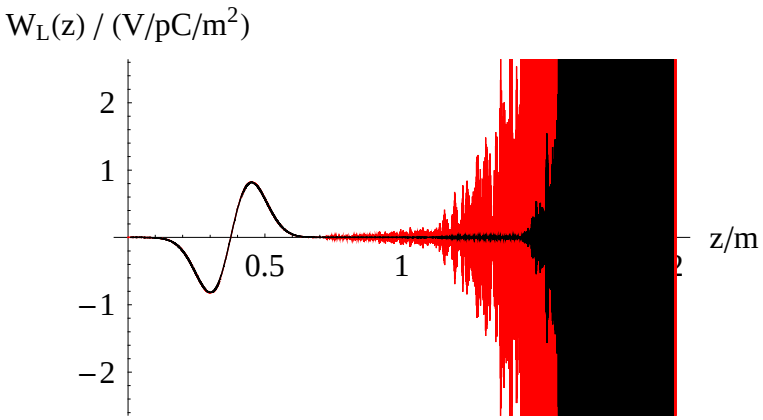
How many points in this wake?

```
In[90]:= Length[sampleWakeData]

Out[90]= 10001
```

A moving average only takes us so far ...

```
In[82]:= ?MovingAverage

          MovingAverage[datalist, r] smooths datalist
             using a simple r-term moving average.  Each element in the
             result is the average of r elements from datalist. More...
```

```
In[91]:= DisplayTogether[rawPlot,
          ListPlot[MovingAverage[sampleWakeData, 100], PlotJoined → True]
         ]
```




```
Out[91]//Short=
      - Graphics -
```

Anyway the point is that we have access to many numerical techniques and can choose or implement what is appropriate for the physics.

Wake as a continouous function

How to transform this wake into a continuous function:

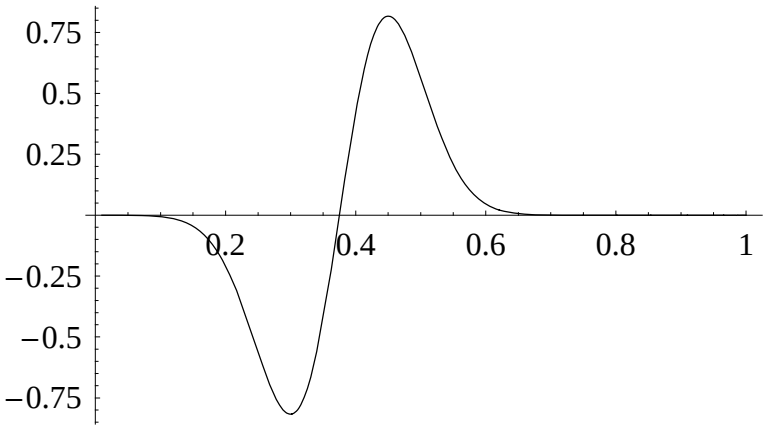
```
In[157]:=
sampleWake = Interpolation[MovingAverage[sampleWakeData, 100]]

Out[157]//Short=
InterpolatingFunction[{{0.0099, 1.9901}}, <>]
```

Now we have a function that we can use it at any z.

```
In[158]:=
sampleWake[.3]

Out[158]=
-0.81693
```

Now we can use Plot instead of ListPlot

```
In[160]:=
Plot[sampleWake[z], {z, 0.01, 1}, PlotRange → All]
```




```
Out[160]=
- Graphics -
```

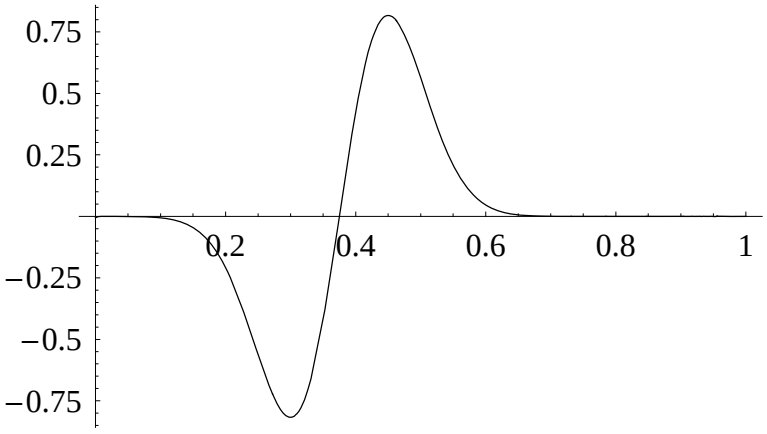

Resampling the wake

How much memory does it use?

In[161]:=
```
ByteCount[sampleWake]
```
Out[161]=
```
198580
```

This is not a big deal but we can show a standard trick to get a smaller but smooth representation of the wake using the adaptive sampling of Plot. Restrict the range for now to avoid the noisy stuff at the end. We go slightly outside the defined range so get some error messages but don't worry.

In[162]:=
```
resampledWakeData = Plot[sampleWake[z], {z, 0, 1.}, PlotRange → All][[1, 1, 1, 1]];
```
```
InterpolatingFunction::dmval :
 Input value {4.16667×10^-8} lies outside the range of data in
   the interpolating function. Extrapolation will be used. More…

InterpolatingFunction::dmval :
 Input value {0.00942492} lies outside the range of data in
   the interpolating function. Extrapolation will be used. More…

InterpolatingFunction::dmval :
 Input value {0.00487935} lies outside the range of data in
   the interpolating function. Extrapolation will be used. More…

General::stop : Further output of InterpolatingFunction::dmval will
   be suppressed during this calculation. More…
```

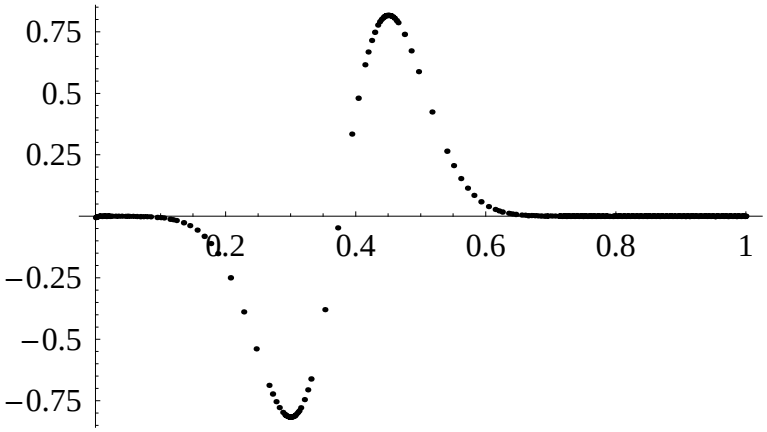


In[163]:=
```
Length[resampledWakeData]
```
Out[163]=
```
390
```

Now the interpolation is based on a set of points chosen so that it will be smooth enough. These points are not evenly distributed:

In[164]:=

```
ListPlot[resampledWakeData, PlotRange → All]
```



Out[164]=

```
- Graphics -
```

Now we can define a new continous function

In[165]:=

```
sampleWake1 = Interpolation[resampledWakeData]
```

Out[165]=

```
InterpolatingFunction[{{4.16667×10^-8, 1.}}, <>]
```

This is much smaller in memory:

In[105]:=

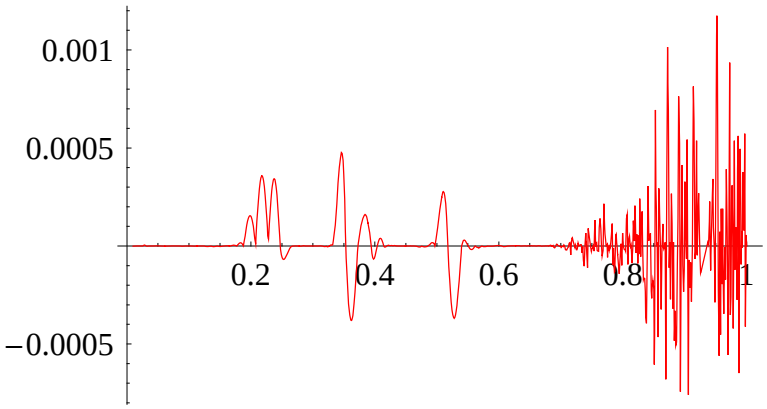
```
ByteCount[sampleWake1]
```

Out[105]=

```
8340
```

Check accuracy

In[167]:=
```
Plot[sampleWake[z] - sampleWake1[z], {z, 0.01, 1.}, PlotStyle → Red, PlotRange → All]
```



Out[167]=
```
- Graphics -
```

This can be refined if it is not good enough.

15 of 19

Analytical models

Natural implementation as *Mathematica* expressions.

Rules to plug in particular parameter values in generic expressions.

16 of 19

Elements and how they occur in the LHC rings

The previous implementation contained information about the occurrences of the impedances in the ring. In principle, each element with an impedance should appear in the MAD description of the LHC. Therefore we should be able to use this as the basic source of information about the occurrences and not duplicate the information in the impedance database. The Madtomma interface to the LHC has already been loaded so we have direct access to the information via its functions.

This is somewhat LHC-specific of course but similar things could be done for other machines.

17 of 19

Example of the LHC RF cavities

Get the optics at the RF cavities:

In[168]:=
```
RFcavityOptics = mfsMember[LHCtwiss["LHCB1"], "KEYWORD", {"RFCAVITY"}];
```

A simple table of interesting quantities

```
In[169]:=
With[{cols = {"NAME", "S", "L", "BETX", "BETY"}},
 TableForm[Transpose[mfsColumn[RFcavityOptics, cols]], TableHeadings → {{}, cols}]]
```

```
Out[169]//TableForm=
```

NAME	S	L	BETX	BETY
ACNCA.D5L4.B1	9955.72	1.1	207.59	159.847
ACNCA.C5L4.B1	9957.72	1.1	203.692	160.834
ACNCA.B5L4.B1	9961.18	1.1	197.122	162.666
ACNCA.A5L4.B1	9963.18	1.1	193.431	163.797
ADTKV.D5L4.B1	9966.04	1.6	188.293	165.503
ADTKV.C5L4.B1	9967.64	1.6	185.481	166.506
ADTKV.B5L4.B1	9970.24	1.6	181.015	168.208
ADTKV.A5L4.B1	9971.84	1.6	178.331	169.3
ACSCA.D5L4.B1	9983.01	1.095	160.941	177.87
ACSCA.C5L4.B1	9984.51	1.095	158.792	179.143
ACSCA.B5L4.B1	9986.	1.095	156.685	180.445
ACSCA.A5L4.B1	9987.5	1.095	154.621	181.777
ACSCA.A5R4.B1	10007.1	1.095	131.488	201.978
ACSCA.B5R4.B1	10008.6	1.095	130.023	203.728
ACSCA.C5R4.B1	10010.1	1.095	128.6	205.507
ACSCA.D5R4.B1	10011.6	1.095	127.22	207.316
APW.5R4.B1	10016.4	2.25	123.11	213.269
ADTKV.A5R4.B1	10023.8	1.6	117.563	223.143
ADTKV.B5R4.B1	10025.4	1.6	116.503	225.369
ADTKV.C5R4.B1	10028.	1.6	114.884	229.058
ADTKV.D5R4.B1	10029.6	1.6	113.951	231.372

These cavities belong to classes, the members of which should have the same impedances. Madtomma has a function that can tell us what they are.

```
In[170]:=
?LHCElementParent
```

```
   LHCElementParent[clel] returns the parent class for an LHC element
     or element class (e.g., any of  "MQXA.1L5", "MQXA", "QUADRUPOLE").
     The parent class of a MAD element class is itself.
```

So we can get the list of parent classes of each RF cavity. The first call of this function builds a database and caches it so future calls will be much faster.

```
In[171]:=
LHCElementParent /@ mfsColumn[RFcavityOptics, "NAME"]
```

```
Out[171]=
{ACNCA, ACNCA, ACNCA, ACNCA, ADTKV, ADTKV, ADTKV, ADTKV, ACSCA, ACSCA,
 ACSCA, ACSCA, ACSCA, ACSCA, ACSCA, ACSCA, APW, ADTKV, ADTKV, ADTKV, ADTKV}
```

The list of different RF cavity types

```
In[172]:=
Union[LHCElementParent /@ mfsColumn[RFcavityOptics, "NAME"]]
```

```
Out[172]=
{ACNCA, ACSCA, ADTKV, APW}
```

Now we can count the number of occurrences of each type

In[173]:=
```
?Frequencies
```
```
   Frequencies[list] gives a list of the distinct elements in
      list, together with the frequencies with which they occur. More...
```

In[174]:=
```
Frequencies[LHCElementParent /@ mfsColumn[RFcavityOptics, "NAME"]] // TableForm
```

Out[174]//TableForm=
```
4      ACNCA
8      ACSCA
8      ADTKV
1      APW
```

If we knew the impedance or wake for each type of cavity, it would now be easy to sum up the contributions using these occurrence counts as weights.

Logically now, one would expect the list of impedance element classe to be a superset of the bottom level of Zbase-Items[]. This should be arranged, I think.

Optics information for impedance calculations

We can also easily do things like calculate the average beta at each element class. Get a table of cavity names, lengths and β_y values, then select just the cavities belonging to a particular class.

In[175]:=
```
With[{elclass = "ACSCA"},
 eldata = Select[Transpose[mfsColumn[RFcavityOptics, {"NAME", "L", "BETY"}]],
   LHCElementParent[First[#]] == elclass &];
 Total[eldata /. {_, l_, b_} → l b]
 ──────────────────────────────────
  Total[eldata /. {_, l_, b_} → l]
]
```

Out[175]=
```
192.22
```

It's easy to generalise that

In[176]:=
```
LHCElementClassAverage::usage =
  "LHCElementClassAverage[seq,elclass,func] returns the length-weighted
    average of the optical function func (e.g. \"BETX\") over the
    elements of an LHC sequence belonging to the element class elclass.";
```

In[177]:=
```
LHCElementClassAverage[seq_?LHCsequenceQ, elclass_String, func_String] :=
 Module[{eldata},
  eldata = Select[Transpose[mfsColumn[LHCtwiss[seq], {"NAME", "L", func}]],
    LHCElementParent[First[#]] == elclass &];
  Total[eldata /. {_, l_, b_} → l b]
  ──────────────────────────────────
   Total[eldata /. {_, l_, b_} → l]
 ]
```

It would not be hard to implement expressions like ("BETX"-"BETY")/"DX" instead of simple strings for func using mfsToRules.

Check the previous result

```
In[178]:=
     LHCElementClassAverage["LHCB1", "ACSCA", "BETY"]

Out[178]=
     192.22
```

And get some more

```
In[179]:=
     LHCElementClassAverage["LHCB1", "MQM", "BETX"]

Out[179]=
     99.5515
```

The caching of the element hierarchy in *Mathematica* makes these calculations quite fast enough.

```
In[180]:=
     LHCElementClassAverage["LHCB1", "MB", "BETX"]

Out[180]=
     85.5717

In[181]:=
     LHCElementClassAverage["LHCB2", "MB", "BETX"]

Out[181]=
     85.6253
```

Actually these are only the averages of the values at the exits of the elements. Probably good enough for instability estimates. But we could improve ...

Conclusions

I have sketched how one might go about implementing an impedance database in *Mathematica*. In my opinion, the advantages would be:

- Natural implementation of impedances which are mathematical functions (sometimes analytical, sometimes numerical, ...).
- Reduced total amount of code-writing and dependences. Functionality closer to physics.
- Based on high quality supported product with no risk of obsolescence. Nothing to compile.
- Familiar environment for many people (but counter-examples exist).
- *Mathematica*'s **`Interpolation`** function allows wakes or impedances given as numerical tables (at discrete points) to be treated as functions defined continuously. So all analytical operations (combination with other expressions, differentiation, etc.) become possible with them.
- Possibility to use existing data structures as-is or to implement new flexible ones (replace fixed format text files with self-describing expressions, rules) and liberate oneself from consideration of data formats.

- Quite straightforward to store input and output files for numerical impedance programs (ABCI, URMEL etc.). Template inputs can have values spliced into them for automatic job submission and tracking on appropriate operating system.

- No need to store optics information in the impedance database as direct access to official LHC optics is provided via Madtomma and can be combined with impedance information in any way desired.

- Direct access via JDBC to all standard databases (or anything that can be found on the Web) for any further information that is needed.

- Direct linking to any Java application (control system, etc.)

- Easy to provide read access to the impedance database via the Web.