Applications of MATLAB to Problems in Quantum Mechanics for Research and Education: Dirac Notation Interpreter

Ladislav Kocbach
Department of Physics, University of Bergen
Allégaten 55, N-5007 Bergen, Norway

Abstract.

Applications of MATLAB to Quantum Mechanics related research and teaching are described. The main application is the Dirac notation Interpreter. In following sections the paper reviews our applications of MATLAB to problems in atomic collision physics, formulated in terms of matrices. These sections are of particular interest to physicists who are considering similar applications in other fields. A special feature of general interest is a demonstration of an easy method to write interpreters of local notations using standard MATLAB functions.

1. Introduction.

This contribution describes applications of MATLAB to quantum mechanical problems which are formulated in terms of matrices. A special feature of general interest is a demonstration that one can quite easily write small specialized command interpreters using MATLAB's character-string functions and MATLAB's command `eval`. In this paper, I first shortly review the Dirac notation as used in Quantum Mechanics. Then I describe the Dirac notation interpreter in MATLAB and discuss mostly the educational aspect of the work. In Technical notes I discuss shortly how this interpreter works and how other interpreters can be constructed. In order to illustrate how this idea emerged and also for the sake of general information, a more detailed presentation of our projects is included in this paper.

It should be well known that a large class of quantum mechanical problems leads to matrix formulation. Our work is related to theory of atomic collisions, but similar problems are found in parts of nuclear physics, optical physics, quantum chemistry etc. We have used MATLAB for example for analysis of problems with non-orthogonal basis. In course of this work, an idea emerged to write a simple interpreter enabling us to enter the statements in easier understandable form. Further developement of these ideas resulted in the above mentioned interpreter package which I hope will be useful for teaching quantum mechanics.

Turning to applications of computers to teaching natural sciences, let us remark that the excellent abilities of the modern integrated systems (including MATLAB, and e.g. Mathematica and Maple) are often difficult to exploit efficiently in teaching. This is caused by specialized command languages and special syntax rules. It takes too much time to learn the language before a simple concept can be demonstrated. This project shows that it is possible to develop interpreters in the command language, which interpret a syntax presumably known to the students. Because of MATLAB's excellent character-string processing repertoir, the interpreter was developed relatively easily.

We also note that the interpreter idea can be extended or adapted to other problems, for example to define vectors (arrays) with vectors as elements (indexed vectors) or multidimensional arrays. This is also shortly described.

2. Dirac Notation.

Dirac's formulation of Quantum Mechanics is based on abstract Hilbert space, which for problems in matrix formulation leads to a set of simple replacements rules.

A matrix H corresponds to an abstract operator $\hat{H}$ and a column vector a corresponds to an abstract vector $|a\rangle$. The eigenvalue equation

$$\begin{bmatrix} H_{11} & H_{12} & \dots & H_{1n} \\ H_{21} & H_{22} & \dots & H_{2n} \\ \dots & \dots & \dots & \dots \\ H_{n1} & H_{n2} & \dots & H_{nn} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ . \\ a_n \end{bmatrix} = E_a \begin{bmatrix} a_1 \\ a_2 \\ . \\ a_n \end{bmatrix}$$

is written in Dirac notation as

$$\hat{H} \mid a \rangle = E_a \mid a \rangle \qquad (1)$$

The scalar product in matrix notation

$$\begin{bmatrix} u_1^* & u_2^* & \dots & u_n^* \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ . \\ v_n \end{bmatrix}$$

becomes in Dirac notation

$$\langle \; u \mid v \; \rangle \tag{2}$$

Very important for quantum mechanics is the so called general matrix element, in Dirac notation

$$\langle \; u \, | \, \hat{V} \, | \, v \; \rangle \tag{3}$$

is in terms of matrices and column vectors

$$\begin{bmatrix} u_1^* & u_2^* & . & u_n^* \end{bmatrix} \begin{bmatrix} V_{11} & V_{12} & . & V_{1n} \\ V_{21} & V_{22} & . & V_{2n} \\ . & . & . & . \\ V_{n1} & V_{n2} & . & V_{nn} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ . \\ v_n \end{bmatrix}$$

The mathematical and physical basis of the Dirac formalism is explained in many texts on Quantum Mechanics, e.g. in [1]. Though developed for quantum mechanics, it could be used in many other applications of linear algebra.

3. The Interpreter.

The Dirac interpreter enables the user to enter the Dirac expressions and have them evaluated by normal MATLAB matrix operations. We have chosen to show the corresponding MATLAB expression before evaluation. Having declared and defined $|a>$ and $|b>$ as two dimensional vectors, we can enter e.g. linear combinations and even assignments (the constants are $c1 = 0.8; c2 = 0.6;$)

```
Dirac :     |a>
MATLAB:   -->     a
a =
     1
     0
Dirac :     |b>
MATLAB:   -->     b
b =
     0
     1
Dirac :     |c> = c1 |a> + c2|b>
MATLAB:   -->      c=c1*a+c2*b
c =
    0.8000
    0.6000
Dirac :     |d> = c2 |a> -  c1|b>
MATLAB:   -->      d=c2*a-c1*b
d =
    0.6000
   -0.8000
Dirac :     <c|d>
MATLAB:   -->     (c)'*d
ans =
     0
Dirac :      <c|c>
MATLAB:   -->      (c)'*c
ans =
     1
```

Projection operators are often a difficult topic. In Dirac notation, using $P = |b>< b|$

```
Dirac :   | b > < b |
MATLAB:   -->     b*(b)'
ans =
     0     0
     0     1
```

We can e.g. easily demonstrate that $P^2 = P$

```
Dirac :   ( | b > < b | )( | b > < b | )
MATLAB:   -->     (b*(b)')*(b*(b)')
ans =
     0     0
     0     1
Dirac :   | b > < b| b > < b |
MATLAB:   -->     b*(b)'*b*(b)'
ans =
     0     0
     0     1
```

The applications to teaching and demonstrations are countless. It should be also mentioned that the interpreter understands and performs the exponential of a matrix (replaces `exp()` by `expm()`), and the functions `sin(x)`, `cos(x)` and `sqrt(x)`.

4. Orthogonalization example.

In this section we show how the well known orthogonalization procedure can be formulated using Dirac notation with projection operators. In the listing are the commands of the example, starting by defining the three nonorthogonal states |a> |b> |c> .

```
D: |a> |b> |c> |u> |v> |w> [U]
M: a=[1;1;0]; b=[1;0;0]; c=[0.3;1;1];
Dirac :   M: U=[1 0 0;0 1 0; 0 0 1] %Unit
U =
     1     0     0
     0     1     0
     0     0     1
Dirac :   |a> = |a> / sqrt(<a|a>)
Dirac :   |b> = |b> / sqrt(<b|b>)
Dirac :   |c> = |c> / sqrt(<c|c>)
Dirac :   % Orthogonalization
Dirac :   |u> = |a>
Dirac :   |v> = ( U  - |u><u|) | b >
Dirac :     |v> = |v> / sqrt(<v|v>) % Norm
Dirac :   |w> = ( U - |u><u| - |v><v| ) |c>
Dirac :     |w> = |w> / sqrt(<w|w>) % Norm
% Tests: (shortened table only); Original:
   <a|b>     <a|c>     <b|c>
   0.7071    0.6359    0.2075
% The orthonormal
   <u|v>     <u|w>     <v|w>
     0         0         0
```

Commands are followed by a table constructed from parts of the responses (the MATLAB forms are omitted). The table shows that the new set |u> |v> |w> is orthogonal, while the original set |a> |b> |c> was not. This example illustrates the possible use of the interpreter for quite complex tasks.

5. Technical Notes.

How does the interpreter work? The main part is a MATLAB function called parse.m. This function translates a string of characters representing a Dirac command into a new string with standard MATLAB syntax. For example

```
>> parse('  <x| M |y> ')
   ans =
       (x)'*M*y
>> parse('( <a| + <b|  ) ( M + N ) |c> ')
  ans =
      ((a)'+(b)')*(M+N)*c
```

The string returned can be evaluated by MATLAB. To do this in a comfortable way, we have a function which sets up the Dirac enviroment. When typing Dirac, the control is taken over by Dirac.m All that we type is then input to this enviroment. This enviroment has been revised many times. One of its functions is to take care of the work done. In the normal version, Dirac copies all the commands into a file. Typing 'H:' (each of the command tokens must be followed by a colon) inside Dirac brings the following help information:

```
Dirac : H:
H:   help
M:   matlab line (execute a matlab line)
D:   Declare a vector    vec :   |vec>
         or an operator   op :   [op]
         or a constant    c1 :   (c1)
     example:
     Dirac  : D: |a> |psi> [L] [Xz] (c1)
     ..declares Dirac vectors    a,psi
                    operators  L, Xz
                    constant   c1
G:   show the global objects
     example:
     Dirac  : G:
     DECLARED :  |a> |b> |psi>
                 [L] [Xz] (c1) (c2)
$    exit Dirac
     a new call Dirac preserves the
     global names and their values
General examples:
e:   lists the 2-components examples
E:   lists the 3-components examples
```

Naturally, the detailed behaviour can somewhat change in the future versions, the described state is end of september 1995. The simplest version of Dirac.m, would look about like this

```
 % simplest possible version of Dirac.m
 stopval=0
 while stopval==0
   InputStr=input('Dirac :    ','s');
   if InputStr(1) =='$' stopval=1; end
   OutStr=parse(InputStr,0);
   fprintf(1,'MATLAB:   -->    %s\n',OutStr);
   eval(OutStr);
 end   % while
```

In fact, the present version of Dirac.m is about one hundred lines and it does many other things, as e.g. passing some of the command strings directly to MATLAB, keeping track of assigned variables etc., as the help text indicates.

Inner working of parse. I think that an appropriate name for it is an heuristic interpreter. The rules for Dirac notation itself can be easily spelled out, but in combination with different uses of parantheses, functions and operators the set of translating rules gets large. Therefore, I keep the parse.m in its original style, with a small set of replacement rules and various post-processing patches. In its present version, the function has about 250 lines, including some comments. Function parse.m is a character string function and it uses the following character string manipulating standard MATLAB functions:

```
% setstr()  assign ASCII value to a character
      QUOT=setstr(39);
% strrep()  replace a substring
      teststr=strrep(teststr,'exp','^') ;
% length()  Length of a string, for loops
      nl = length(teststr)
% findstr()   returns a list of occurences
      findstr(teststr,'<')
% splitting strings
      yyy=teststr(1:nl-3)
% setting strings together
      yyy=[ yyy '(' NaME ')' QUOT  ]
```

The simplest first test version, which could only do the scalar product <a|b> used only the strrep() and returned the new string. The present version can handle all the combinations of signs which ocurred to the author. In fact, recently a problem with parantheses has been encountered and the repair of the inconsistency contributed some ten new lines of code.

For teaching and demonstration purposes the standard MATLAB output is not always suitable. At present, we are thus also working with formatting routines, as an example this is a printout of a 3×3 matrix

```
 (2.         ; 0.          ; 0.         )
 (0.         ; 1.          ; 1.+    1.i )
 (0.         ; 1.-    1.i  ; -1.        )
```

Concluding this section, I would like to repeat that it is in principle quite easy to write an interpreter using the string functions mentioned in the above listing. The interpreter (or perhaps parser) can then be used via a function shown in the listing Simple Dirac or directly. Unfortunately, only single line codes can be performed in this way, since both input() and eval() understand newline-character as a terminator of the string. On the other hand, there seems to be no limitation on the length of the string which can be executed by eval() . An interpreter can pack longer commands in the same way as the following example shows

```
>> % a long program in 4 strings
>> L1='X=zeros(10); ';
>> L2='for k1=1:10 if k1>3 ';
>> L3='for k2=1:10 X(k1,k2)=k1*k2; ';
>> L4='end; end; end; ';
>> % packed into 1 line
>> eval([ L1 L2 L3 L4 ] );
>> % and this checks it
>> X(4,5)+X(3,5)
ans =
     20
```

6. Time-dependent Schrödinger equation for description of atomic collisions.

The approach to atomic collisions implied in this section is called semiclassical, referring to the fact that only the electrons are treated as quantal particles, while the atomic motion is simulated by a classical trajectory. This leads to time-dependent Schrödinger equation (eq. (5) below). The matrix formulation of this problem arises from expansion of the unknown electron wavefunction $\mid \psi(t) \rangle$ in a set of basis functions $\mid \phi_i \rangle$, much in analogy with Fourier series or expansions using orthogonal polynomials

$$\mid \psi(t) \rangle = \sum c_i(t) \mid \phi_i \rangle \tag{4}$$

The unknown quantities to be found are the expansion coefficients, which form a vector. In this formulation, the time-dependent Schrödinger equation

$$i \frac{\mathrm{d}}{\mathrm{dt}} \mid \psi(t) \rangle = \hat{H}(t) \ \mid \psi(t) \rangle \tag{5}$$

is replaced by a set of coupled differential equations, which are conveniently expressed by matrix notation

$$i \frac{\mathrm{d}}{\mathrm{dt}} \begin{bmatrix} c_1 \\ c_2 \\ . \\ c_n \end{bmatrix} = \begin{bmatrix} H_{11} & H_{12} & \dots & H_{1n} \\ H_{21} & H_{22} & \dots & H_{2n} \\ \dots & \dots & \dots & \dots \\ H_{n1} & H_{n2} & \dots & H_{nn} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ . \\ c_n \end{bmatrix}$$

Normaly, the basis functions would be orthonormal. In many of the physical problems we study, the natural sets of basis functions are not orthogonal. There are currently several possible approaches to this problem, using matrix inversion or pre-orthogonalization of the basis set. MATLAB has proven to be a very useful tool for debugging the FORTRAN-based codes and in particular, to analyze the actual conceptual problems. Some details of the collision codes can be found e.g. in [2] or [4].

7. General Comment on Collision Studies.

For the atomic collisions calculations, a set of quite large FORTRAN codes exists and new codes are under developemnt. Here it is described how MATLAB has recently been applied in the process of developement and debugging. In these applications, MATLAB's matrix handling and its integrated graphical abilities are very valuable and cannot be matched by any other approach. On the other hand, other modern tools are also extremely useful in the discussed work.

We are applying REDUCE with GENTRAN to check the correctness of algebraic relations in symbolic calculations and outputing directly functioning FORTRAN code, which is integrated in the system. Here the combination REDUCE and GENTRAN are unmatched. Mathematica has been useful for our work because of its large built-in library of physics related functions.

From this we can see that the amount of time spent in mastering to a sufficient degree the details of the various mentioned systems is large and that a cross-interpreter and information systems would be of great value for efficient application of these powerful tools.

8. Nonorthogonal basis sets.

We describe here another method to treat the nonorthogonal basis sets, which seems to be well known in Quantum chemistry [3]. It is based on a treatment of the overlap matrix, $o_{ij} = \langle \ \phi_i \ \mid \phi_j \ \rangle$

$$\begin{bmatrix} o_{11} & o_{12} & \dots & o_{1n} \\ o_{21} & o_{22} & \dots & o_{2n} \\ \dots & \dots & \dots & \dots \\ o_{n1} & o_{n2} & \dots & o_{nn} \end{bmatrix} \tag{6}$$

The overlap matrix O has eigenvalues $O_1, O_2, ..., O_n$ and its eigenvectors are arranged as columns in matrix S. Defining a matrix U

$$U = \begin{bmatrix} (O_1)^{-1/2} & 0 & \dots & 0 \\ 0 & (O_2)^{-1/2} & \dots & 0 \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & O_n^{-1/2} \end{bmatrix} \tag{7}$$

a transformation matrix T can be constructed

$$T = SU \tag{8}$$

It is easy to see that

$$T^{+}OT = US^{+}OSU = \mathbf{1} \tag{9}$$

so that the (scaling) transformation T transforms all the relevant operators from the original basis to a new basis, which is orthonormal.

This is all well as far as the functions (i.e. vectors) of the basis remain abstract objects and we work only with the overlap matrix and other relevant operators. In an application, however, we might also work with the objects themselves (e.g. to plot their representation), and they might be e.g. $\mathcal{N}$-dimensional vectors (such that their dimension $\mathcal{N} >> n$). Unfortunately, MATLAB cannot place such general objects in a vector. We have written a demonstration routine, exploring possible solutions of these problems (see also section 11. or the next section, which includes plots).

9. Quantum Chemistry inspired Example.

An extreme case of non-orthonormal basis is the use of so–called Gaussian orbitals in Quantum Chemistry and in some physical applications. Here the basis

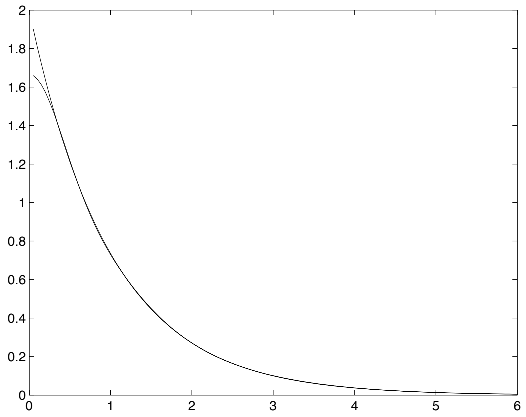

Figure 1: Comparison of the combined Gaussian with the hydrogen ground state

functions (or states) are the functions

$$| \alpha_i \rangle \rightarrow e^{-\alpha_i r^2}$$

which are far from orthogonal with respect to the scalar product defined as

$$\langle \alpha_i \mid \alpha_j \rangle \rightarrow \int_0^\infty e^{-\alpha_i r^2} e^{-\alpha_j r^2} r^2 dr$$

The expressions for all the relevant matrix elements are well known analytically and matrices for overlap and the atomic total energy (hamiltonian) are thus easily constructed. The alternative orthogonalization procedure (eq.9) has been applied in terms of a small set of MATLAB functions. The figures 1 and 2 show a typical output for a randomly chosen set of α_i. Motivation for this work is to explore the procedures for choosing sets of orbitals and understanding the procedures used in quantum chemistry. It will also be used in the course of Atomic and Molecular Physics. The MATLAB functions are available as mentioned in the Conclusion.

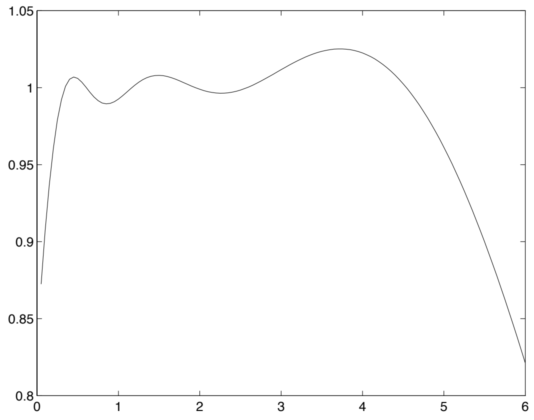

Figure 2: Ratio Gaussian to exact as seen in Fig.1

10. Collision Code Debugging .

Our general interest is to investigate the two mentioned orthonormalization procedures with respect to questions which will become clear at the end of this section. In the theory of atomic collisions, especially those directed towards the studies of electron transfer, the basis states for the electron are located on both atomic centers. When the atomic centers are far apart, the overlaps are small and the basis is approximately orthonormal, if each of the two subsets on each center were orthonormal. As the two atoms approach each other, the two subsets gradually overlap and for the smallest distances the effective dimensionality of the basis might become smaller (approaching the dimensionality of one subset). The small subspace representing the difference between the two subsets is 'blown up' by the normalization and this can lead to numerical problems. This can be studied by investigating the determinant of the overlap matrix. For well separated atoms it approaches one (if the two subsets are internally orthonormal for each center), but as the atoms approach each other, the determinant approaches zero.

All the matrix elements involved in the realistic atomic collision calculation are quite complicated mathematical objects and they were thus evaluated by the collision FORTRAN program (ref. [4]). The overlap matrices were dumped as one large rectangular matrix and entered into MATLAB. They were then investigated using simple MATLAB functions.

In particular, plotting the determinant of the overlap matrix eq. (6).

$$D(R) = \det(O(R))$$

as function of the interatomic distance R, we were able to detect the reason for numerical problems in collision calculations.

In short, it can be shown that $D(R)$ cannot cross zero, it can at most have a zero minimum. It turned out that some of the numbers were not evaluated but set precisely to assumed, theoretically correct values. After replacing this by evaluation, the resulting values remaind positive. This is connected with the fact that the states are nearly linearly dependent (overlaping) and calls for further detailed investigation of the calculational approach, the details are however not interesting in this discussion.

The problem of gradually overlapping basis subsets is however quite general and the applications of symmetric orthonormalization and elimination of unphysical states is of broader interest. The work on these aspects is done together with A. Dubois and a paper is in preparation [5] .

11. Multidimensional Array Interpreter.

The technical notes included in our MATLAB release contain a suggestion how to treat multidimensional arrays. It is based on index pointing into a rectangular array.

In one of our applications we needed to have an array of vectors, but it could easily be a vector of matrices. Here we show how it is possible to treat it by interpreting a notation. In this example, the interpreted string is fed directly to `eval`.

```
>>%  These must be written explicitely
>>  c_1=v(:,1); c_2=v(:,2);
>>  c_3=v(:,3); c_4=v(:,4); c_5=v(:,5);
>>%  Here is the same using ToVector()
>>   for vind=1:5
       eval(ToVector('c_','v',vind));
     end
>> vind=4;
>> ToVector('c_','v',vind)
ans =
        c_4=v(:,4);
```

Listing of function ToVector.m

```
function os= ToVector( vname, mname, ki)
% translates for vector-array
% outputs (ki=4)
% vname4 = mname(:,4);
 os = [ vname sprintf('%d',ki) '='];
 os = [ os  mname '(:,' ] ;
 os = [ os sprintf('%d',ki) ');' ];
```

This enables us to use the vectors

```
c_1,... c_5...
```

as if they were indexed vectors. In the simple example here only the assignment is shown, but a general interpreter can be written, using for example the trick to pack longer commands into a single string as shown at the end of section 5.

12. Conclusion.

This paper described the applications of MATLAB. It should, however also be mentioned that other modern integrated systems are also used in this work. These include REDUCE, Mathematica and Maple. For numerical intensive tasks MATLAB is unique. For the educational applications, along the lines of the Dirac notation interpreter, the mainly algebraic systems may be even more suitable. However, MATLAB's internal representation of vectors, matrices and character strings shortened the way from the idea to the first realization to several hours.

The files and documentation for the Dirac interpreter can be found in the World Wide Web at the address

`http://www.fi.uib.no/AMOS/matlab/`

Also some of the other mentioned functions can be found there. If necessary, the files can be transfered in any other way on request to the author.

Acknowledgements.

I would like to thank J. P. Hansen, A. Dubois and S.E. Nielsen for the impulses from the collaboration on atomic physics codes, K. Børve for clarifying discussions on the methods of Quantum Chemistry, and my son Jan Kocbach for first introduction to MATLAB.

References

[1] E. Merzbacher: Quantum Mechanics (Wiley International Edition)

[2] J. P. Hansen and K. Taulbjerg, Comp. Phys. Comm. **51**,317 (1988) A preorthonormalization procedure for coupled channel problems

[3] K. Børve (1995), private communication

[4] J. P. Hansen and A. Dubois Comp. Phys. Comm. **67**, 456 (1992) Procedures for analytical and numerical calculation of Coulombic one- and two-centre integrals

[5] A. Dubois and L. Kocbach, in preparation (1995)