

USING SYMBOLIC ALGEBRA FOR THE GENERATION OF ORBIT SIMULATION CODES FROM HAMILTONIANS

Andreas Adelman* and Stefan Adam,
Paul-Scherrer-Institute, CH-5232 Villigen, Switzerland

Abstract

As a part of our new research activity aiming at a detailed understanding of space charge effects in ring cyclotrons and in the corresponding injection beam lines at the Paul Scherrer Institute we are currently developing a three dimensional space charge simulation code.

The use of the Hamiltonian formalism, in combination with a symbolic algebra system (MAPLE), enables us to carry out the entire modelling on a high level. MAPLE easily handles the elaborate derivatives required to form the equations of motion and then casts them into FORTRAN. The subsequent embedding of the FORTRAN code into the MATLAB package is also handled automatically by the procedures of our framework. MATLAB is well suited for running small simulations followed by various post processing activities including graphics.

In order to test this framework, two model cases have been chosen: the double focusing spectrometer and the motion of a single particle in a stable charged cloud. The successful processing of these and some further small model problems encourages us to apply this framework to produce code for the large scale simulation.

The development steps from the Hamiltonian to the FORTRAN subroutine and the resulting simulations are shown for two model cases, as well as the general MAPLE and MATLAB procedures we have developed for this purpose.

Keywords: space charge, symbolic algebra, code generation, Hamiltonian system

1 HISTORY AND MOTIVATION

The acronym FORTRAN stands for **FOR**mula **TRAN**slation. The computer language FORTRAN was originally developed by John Backus in 1954 and first released 1957 by IBM [1] [2]. FORTRAN was one of the first high level computer languages in which the programmer specifies the task of the computer in terms of human readable text. This resulted in programs being easier to read, understand and debug. It also allows the programmer to concentrate on the actual problem, rather than getting diverted into details of the underlying computer architecture. If we look more closely however, the formulas written in the natural language of mathematicians and physicist still have to be translated into an artificial language understood by the computer. This need for an additional transforma-

tion step is present for all other programming languages. For a long time this has been a major motivation in the search for a direct way to translate a mathematical description into computer code [6] [8]. The advent of symbolic algebra systems opened the option of entering problems in the mathematical language. With the option to automatically generate code from formulas the two diverging languages approached each other.

We extend that further, towards a automated generation of a small scale orbit simulation starting from the Hamiltonian. The benefits of this are:

1. Time efficient generation of orbit simulation code
2. Quality of the resulting code with respect to correctness
3. Test of the physical and mathematical model, based on analytically solvable examples

2 SYMBOLIC MANIPULATION

2.1 Problem Frame

Following the definition of a computer experiment [10], we introduce two categories: **1. Large Scale Simulation** and **2. Small Scale Simulation**. The large simulation will meet our scope to simulate the space charge effects of huge a number of particles in a complicated accelerator structure. Given the complexity of the large problem, a successive approach using several small steps is appropriate. Figure 2.1 shows how the two categories interact. After a few iterations of the *loop*, comprising the Small Scale Simulation, the mathematical model is expected to be sufficiently validated to justify the use of the corresponding code in the large scale simulation (see *is used by* in Figure 2.1).

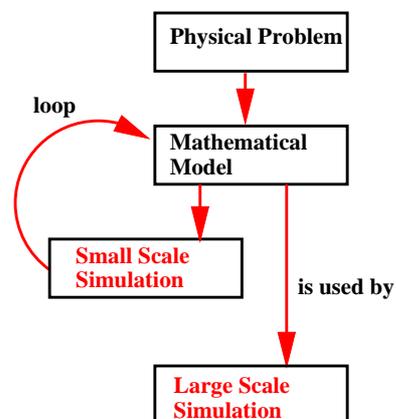


Fig. 2.1 The Frame of Discussion

* Corresponding Author: Andreas Adelman (Andreas.Adelmann@psi.ch)

The description of the problem starts with its Hamiltonian

$$H(\mathbf{q}, \mathbf{p}; t) \quad (1)$$

from where we can get the corresponding Hamiltonian equations

$$\frac{dq_i}{dt} = \frac{\partial H}{\partial p_i}, \quad i = 1 \dots 3 \quad (2)$$

$$\frac{dp_i}{dt} = -\frac{\partial H}{\partial q_i}, \quad i = 1 \dots 3 \quad (3)$$

which then lead to the equations of motions.

2.2 Modelling Stages for Orbit Simulations

We are interested in the trajectories of particles guided by various forces, such as: external forces, collective forces (space charge), etc.

$$H(\mathbf{q}, \mathbf{p}; t) = H_{ExternalForces} + H_{SpaceCharge} \quad (4)$$

The effect of a guiding magnetic field on charged particles is best expressed with the vector potential:

$$\mathbf{A}(\mathbf{q}; t). \quad (5)$$

In the sequel we neglect space charge ($H_{SpaceCharge} = 0$) and define the unified notation:

$$\mathbf{y} = (\mathbf{q}, \mathbf{p})^T. \quad (6)$$

With this, the equation of motion form a system of first order differential equations,

$$\frac{dy}{dt} = F(\mathbf{y}, t) = \left(\frac{\partial H}{\partial p_i}, -\frac{\partial H}{\partial q_i} \right)^T \quad (7)$$

which we finally have to solve.

3 THE SIMULATION SHELL

Assume one has (7), out of a symbolic algebra system. To move on and build a small scale simulation, one might consider three alternatives:

1. remain in the symbolic algebra system to solve (7) [7]
2. automatic transfer of (7) into a programming language and coding the simulation (see Figure 2.1).
3. **transfer of (7) into a programming language and joining this code to a numerical simulation package.**

To achieve our goal of validating the mathematical and physical model with minimal programming effort (in the sense of 3rd generation programming languages), we follow the third alternative.

We have chosen the MAPLE package [5] for all symbolic manipulations, and use the extended capabilities of MACROFORT [9] for code generation and apply MATLAB [11] for the numerical part.

As a starting point we have to build symbolic expression for the Hamiltonian (4). With symbolic differentiation, MAPLE can evaluate the equation (7) for the given Hamiltonian. The resulting expressions are **automatically translated** into FORTRAN subroutines.

In order to use external Fortran subroutines within the MATLAB environment, we have to build an intermediate glue layer, the so-called MEX-Interface [11]. The Mex-Interface uses a gateway subroutine in addition to the original external function (see Figure 3).

This glue layer itself is fully defined by the signature of the external function, denoted as F and A in our example shown in figure 3.

With this procedure, we generate functions which represent F in (7) and (A) from (5). Those can then act like MATLAB built-in functions.

The MATLAB environment allows to solve (7) numerically using built-in, or external solvers for the system of ordinary differential equations (ODE's), as the problem requires.

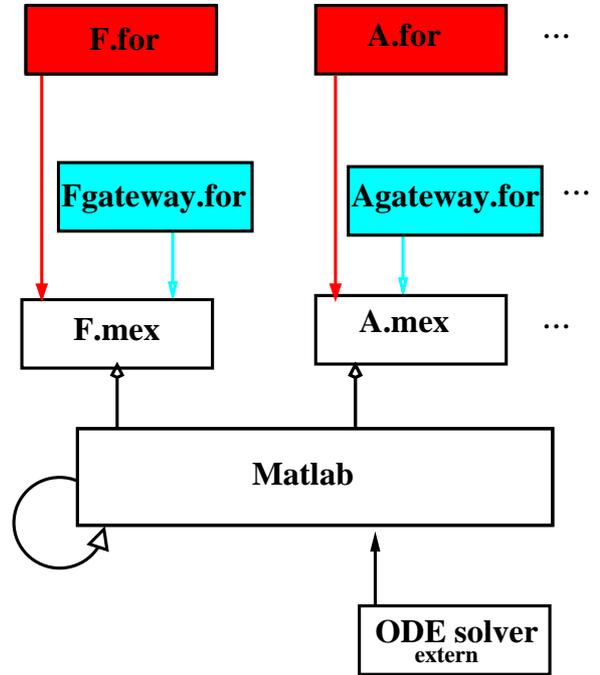


Fig. 3 The Simulation Shell

4 EXAMPLES

Two examples will be used to show in detail the steps from the abstract formulation towards a running small scale simulation.

4.1 The double focusing Spectrometer

The mathematical formulation translates one to one into MAPLE syntax, as can be seen below.

The Hamiltonian for a particle in an external magnetic field is given by Barut [3]:

$$H = c \cdot \sqrt{(\mathbf{p} - e_0 \mathbf{A}(\mathbf{q}; t))^2 + m_0^2 c^2} \quad (8)$$

where \mathbf{A} is the vector potential. The case in consideration does not contain electrostatic fields, therefore we omit those terms.

To obtain the characteristics of the double focusing spectrometer [13], a rotationally symmetric, radially decreasing field has to be chosen, with a radial component of:

$$B_r = B \cdot R^n \text{ and } R = \sqrt{(q_1^2 + q_2^2)}, \quad (9)$$

where the field index n amounts to 0.5. This field has the corresponding vector potential with the azimuthal component A_θ that can be obtained by algebraic manipulation:

$$A_\theta = -\frac{1}{R} \cdot \int R \cdot B_r \cdot R^{n_{Field}} \cdot dR + \frac{q_3^2}{2} \cdot \frac{\partial}{\partial R} (B_r \cdot R^{Field}) \quad (10)$$

4.1.1 Symbolic Algebra Part

Assuming the expressions for (10) with n as parameter are stored as elements of the matrix A , the Hamiltonian can be written in the MAPLE algebra system as follows:

$$H := c * ((p[1,1] - e_0 * A[1,1])^2 + (p[1,2] - e_0 * A[1,2])^2 + (p[1,3] - e_0 * A[1,3])^2 + m_0^2 * c^2)^{(1/2)} :$$

This corresponds to the definition of the Hamiltonian (4) where space charge is neglected.

To obtain the equations of motion (7), a set of derivatives has to be formed, entering the MAPLE commands:

```
for i to DIM do
  d_eqs[i] := diff(H, Y[eval(i+DIM)]);
  d_eqs[eval(i+DIM)] := diff(-H, Y[i]);
od;
```

The parameter DIM defines the dimension of the system to be studied.

The generic code generation procedure gen_F , contains an additional parameter $NPART$, the number of particles used in the simulation.

```
gen_F : proc(DIM, NPART)
  local j, pg;
  0  pg := [];
  1  pushe(['declaref', 'doubleprecision',
  2      [T, Y[2*DIM*NPART], YPP[2*DIM*NPART]]], 'pg');
  3  pushe(['commonf', 'GLOBAL', GLOBALLIST], 'pg');
  ...
  5  for j from 1 to 2*DIM do
  6      pushe([equalf, YPP(j), d_eqs[j]], 'pg');
  od;
  ...
  9  genfor(pg);
end;
genfor(gen_F(DIM, PART)):
```

Fig. 4.1.1 Fragment of the code generation procedure

In this procedure we make use of functions provided by the MACROFORT package [9].

With the MACROFORT commands *pushe* (line 1 - 6) the list *pg* is filled with information related to declarations and statements of the code to be generated. Finally, this information is translated into the FORTRAN subroutine *Ffor* upon the command *genfor* (on line 9). The FORTRAN sub-

routine *Afor* is obtained in a similar way, therefore the detailed description is omitted here.

4.1.2 The Numerical Part

In the MATLAB environment we do the numerical simulation, which starts by defining initial conditions for position and momentum of the particle. To calculate these initial

conditions the vector potential function A is used to get the mechanical momentum from the canonical one.

```

n      = 0.5;

q10   = 0.0;
q20   = 0.5;
...
p0    = 0.0;
...
Ainit = A(q10,q20,q30,n);

p10   = -p0 + Ainit(1);
p20   = 0.0 + Ainit(2);
...
y0    = [q10 q20 q30 p10 p20 p30 ]';

```

We then use the Runge Kutta Fehlberg (4th order) integrator from MATLAB to integrate the ODE system.

Note that in these samples of Matlab code the functions A and F cannot be distinguished from original MATLAB built-in functions.

```
[t1,y1]= ode45('F',TSPAN,y0,options);
```

Finally, one can use a variety of graphical and other post processing functionalities available in MATLAB.

```
plot3(y1(:,1),y1(:,2),y1(:,3),'g');
```

The plot below shows the characteristics of the double focusing spectrometer: horizontally and vertically diverging orbits meet at the azimuthal angle $\sqrt{2} \cdot \pi$ from the start.

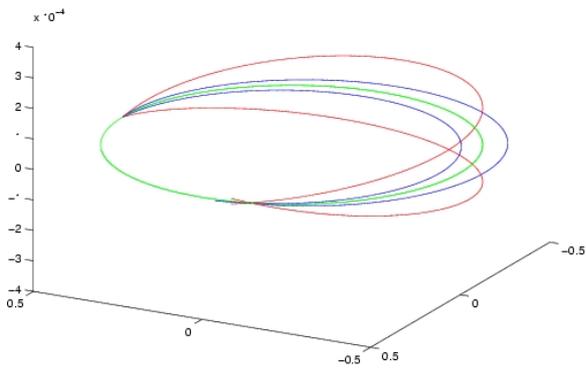


Fig. 4.1.2 Particle Trajectories in a Double Focusing Spectrometer

4.2 Particles in a Space Charge Cloud

As another test case to verify the outlined procedure for automatically generating orbit simulation code, we consider a

particle moving in a charged cloud. The governing Hamiltonian becomes:

$$H(\mathbf{q}, \mathbf{p}; t) = H_1 + H_2 + H_{SpaceCharge} \quad (11)$$

We then proceed in a way similar to the previous example and obtain results, which are in agreement with [4], see Figure 4.2.

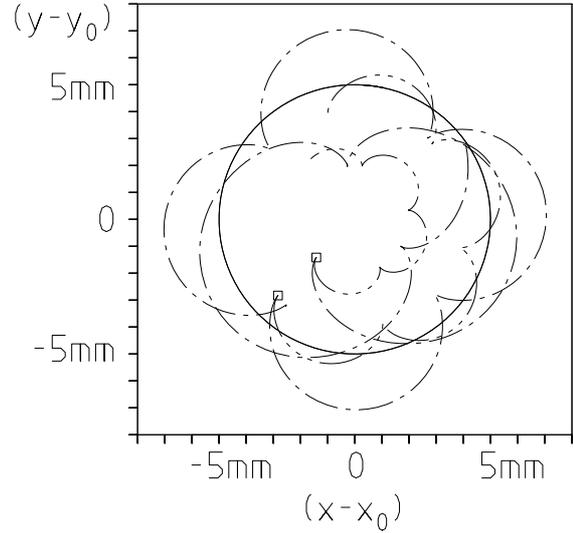


Fig. 4.2 2D Particle Motion inside a Charged Cloud

The generality of the presented scheme, for the automatic generation of orbit simulation code, encouraged us to extend the case of particle motion in a space charged cloud to three dimensions.

5 CONCLUSIONS AND FUTURE

We have successfully established a framework for the automatic generation of orbit simulation code, without using any 3rd generation programming language.

Starting with the formulation of the Hamiltonian in mathematical notation, the error prone tasks of forming derivatives and of coding in FORTRAN are fully handled by MAPLE. The successive inclusion into the MATLAB environment is done by automatic generation of the MEX-Interface. The essential parts of this framework and their interactions are shown in Figure 5.

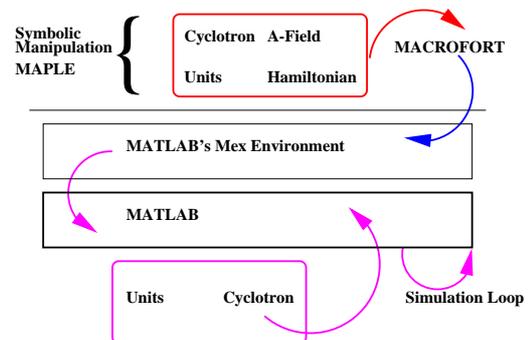


Fig. 5 The Framework of Automatic Codegeneration

This framework is useful for validating physical and mathematical models, by using well-known existing components:

1. MAPLE and MACROFORT
2. MATLAB
3. ODE solvers

Essential software components to be contained in a large scale simulation can be easily produced and tested within this framework:

Equations

⇒ SW-Components

⇒ tested SW-Components

We will further use this approach to generate important code parts for our research project:

The construction of a three dimensional beam simulation code for high intensity proton beams. This code will allow us to study the behaviour of space charge dominated beams at injection into the PSI Injector 2. [12]

We plan the extension of this framework in the following aspects:

- Simulations based on symplectic maps
- Interface to a parallel computing environment
- Rewrite MACROFORT to handle the programming language C

Acknowledgements

We are indebted to the Professors R. Eichler and R. Jeltsch and to Dr. T. Stambach for their support of our project and for many fruitful discussions.

6 REFERENCES

- [1] J. Backus. Can programming be liberated from the von Neuman style? *Communications of the ACM*, 21, Aug. 1978.
- [2] J. Backus. The history of FORTRAN I, II, and III. In R. L. Wexelblat, editor, *History of programming languages. Proceedings of the ACM SIGPLAN conference (Los Angeles, Calif., June 1–3, 1978)*, pages 25–74, New York, NY, USA, June 1981. Academic Press.
- [3] A. Barut. *Electrodynamics and Classical Theory of Fields and Particles*. Dover, 1980.
- [4] A. C. Chasman. Space charge effects in a heavy ion cyclotron. *Nuclear Instruments and Methods in Physics Research*, pages 279–283, 1984.
- [5] B. W. Char, K. O. Geddes, G. H. Gonnet, B. Leong, M. B. Monagan, and S. M. Watt. *First Leaves: Tutorial Introduction to Maple*. Springer-Verlag, New York, 1992.
- [6] J. S. Cohen. The effective use of computer algebra systems. In *Transactions of the Sixth Army Conference on Applied Mathematics and Computing*, pages 677–698, 1989.
- [7] E.S.Cheb-Terrab and H. de Oliveira. Poincare sections of hamiltonian systems. *Phys-Pub Universidade de Estado do Rio de Janeiro and submitted to CPC*, January 1996.
- [8] J. Fitch. Mathematics goes automatic. *Physics world*, 6(6):48–52, June 1993.
- [9] C. Gomez. MACROFORT: A FORTRAN code generator for MAPLE. Technical report, Institut National de Recherche en Informatique et en Automatique, 1990.
- [10] R. Hockney and J. Eastwood. *Computer Simulation using Particles*. Adam Hilger, 1988.
- [11] I. T. MathWorks. MATLAB 5. URL: <http://144.212.100.10/>, 1997.
- [12] PSI-Home-Page. Injector 2. URL: <http://www.psi.ch>, 1998.
- [13] K. Siegbahn. *Alpha-, Beta- and Gamma-Ray Spectroscopy*. North-Holland, 1964.