Modeling and Simulation of Mechanical Systems
— Combination of a Symbolic Computation Tool and MATX —

Tasuku Hoshino and Katsuhisa Furuta
Graduate School of Information Science and Engineering
Tokyo Institute of Technology
2-12-1 Oh-okayama, Meguro, Tokyo 152-8552, JAPAN
thoshino@mei.titech.ac.jp

Abstract

This paper illustrates an efficient way of modeling mechanical systems and performing the numerical simulation, by combinationally using symbolic and numerical computation tools. Since deriving the model and its minimal representation involves symbolic manipulations of equations, it must be handled by the symbolic computation tool. On the other hand, the simulation task requires numerical evaluations of the same object repeatedly; it can be efficiently processed by the numerical tool, especially when the values of objects are immediately available. The key point is how to export the symbolic objects easily into the numerical environment. The authors try to automatically generate source codes of the objects for the numerical tool, and join two environments on the source level. The stabilization of the spherical pendulum is served as an example, and its modeling and the simulation using *Mathematica*/MATX are included.

1 Introduction

In the computer-aided control system design, both symbolic and numerical manipulation tools for equations play important roles. Usually, since a symbolic object has more complex internal data representation than that of a numerical object, symbolic tools can perform several kinds of operation other than numerical evaluation. On the contrary, numerical tools can evaluate numerical objects efficiently, in speed and size. By combinationally using both kinds, flexible manipulation of equations is made possible.

The modeling and simulation of mechanical systems are typical examples where such 'mixed' manipulation of equations is effective. The structural model is systematically given through the Lagrangian formulation with some symbolic operations. Once the parameter values are identified, the model becomes 'fixed' to give a numerical object which is used in the numerical simulation. Recent rapid growth of computing power enables us to handle mechanical systems with many degrees of freedom (d.o.f.). The use of the symbolic tool as well as the numerical tool is getting more important and brings the design task more possibilities. An example of Maple/Scilab combination is found in [2].

This paper deals with the modeling and the simulation of mechanical systems using symbolic and numerical software, focusing the task assignment: the structure is modeled by *Mathematica* and quantitative modeling is done by MATX package [1]. Since MATX provides a compiler as well as an interpreter, the fast execution of simulations of large scale systems becomes possible when the values of objects are immediately available.

It should be noted, that in some cases, the modeling process, the simulation task, and even the control computation can be accomplished without symbolic models. For example, the well-known recursive Newton-Euler algorithm and the unit vector method numerically construct the simulation model; it is also useful in computing the linearization feedback. Nevertheless, since symbolic models work better than the recursive model when the d.o.f. is small and there also exist many cases where symbolic models are necessary. In addition, the symbolic manipulation of equations is required in other cases of controller design: the factorization approach over polynomial matrix ring or rational polynomial matrix field, or the feedback linearization method via differential geometric approach. This paper includes these cases within its scope.

After summarizing a typical modeling process in the next section, the task assignment and exporting the objects are described in section 3. The proposed method will be illustrated with an example on the spherical inverted pendulum [4] in section 4.

2 Models and the modeling

The following three models are typically used in the design process of controllers for mechanical systems:

Identification model: The equations of motion or the energy of the system can be represented as a regression form since they are linear with respect to the dynamics

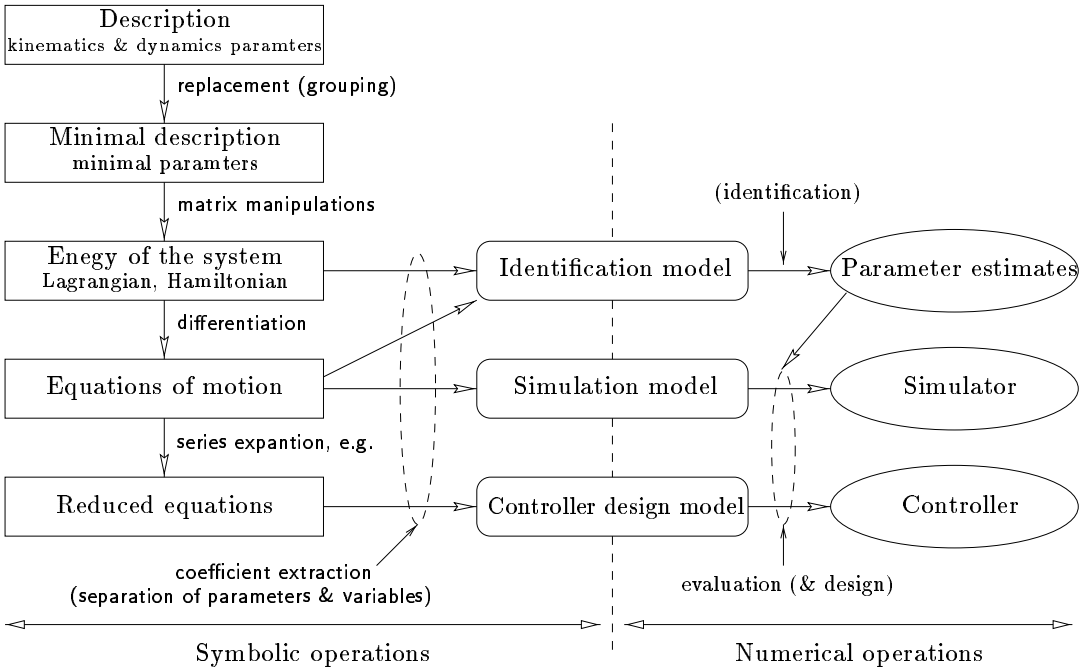

Figure 1: Symbolic and numerical operations in the CACSD for mechanical systems

parameters. Their least-square estimates are obtained based on either regression model of the equations of motions or of the system energy.

Simulation model: The numerical simulation is to solve the initial value problem of the equations of motion via Runge-Kutta algorithm, for example. The nonlinear state equation, which gives time derivative of the state vector is required as the simulation model.

Controller design model: When designing controllers, a reduced or transformed model is often used instead of the original.

Given a mechanical system, the derivation of these models can be summarized as follows.

System description: To formulate the system, description of both kinematics and dynamics are necessary. While the well-known Denavit-Hartenberg (D-H) notation describes the kinematics by attaching local coordinate frames fixed to each body, the modified D-H notation is used hereafter since it has several advantages over the original [3]. The inertial parameters, i.e., the mass, the inertial tensor, and the position of the center of gravity, of each body are measured in each local coordinate. Since when all the parameters above are available, the numerical construction of the model is possible as mentioned earlier, this description gives another alternative 'model' of the system.

Parameter reduction: The parameters above is redundant in the sense that not all of them are identifiable; in order to get the identifiable set (also referred to as the base parameters; consisting of base inertial parameters and friction parameters), the parameter reduction is necessary. The reduction procedure is translated in the parameter space [5]. The reduction corresponds to grouping some parameters, and it can be performed by recursive symbolic replacements of parameters. This reduction also reduces the computational amount.

Energy of the system: The kinematics of the system can be described in terms of the homogeneous coordinates and their transformations. Based on the analysis, the total kinetic energy T and the total potential energy U of the system can be computed. It involves symbolic operations of matrices and vectors, and symbolic differentiations.

When identifying the base parameters using the energy model, the Hamiltonian of the system $H := T + U$ is transformed into the regression model:

$$\Delta H = \Delta \boldsymbol{\phi}_e(\boldsymbol{q}, \dot{\boldsymbol{q}})^T \boldsymbol{a} = W, \tag{1}$$

where $\boldsymbol{q}$ is the generalized coordinate, $\boldsymbol{a}$ is the vector of the base parameters and $\Delta\boldsymbol{\phi}_e(\cdot)$ is the regressor; W is the supplied energy. This transformation, i.e., the construction of $\boldsymbol{\phi}_e(\cdot)$ is done by symbolically extracting the coefficient of each base parameter from H. Based on the regression model, the optimization of the exciting trajectory and the identification experiment are carried out to determine the estimates of the base parameters.

Equations of motion: Using the Lagrange equation, the equations of motion of the system are derived by symbolic differentiations of Lagrangian $L := T - U$:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{q}_i}\right) - \frac{\partial L}{\partial q_i} = Q_i - \frac{\partial R}{\partial \dot{q}_i}, \quad i \in \{1, \cdots N\}, \tag{2}$$

where R is the dissipation energy, Q_i's are the generalized forces, and N is d.o.f. of the system. Equations

(2) can be rewritten into the following form:

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{F}(\boldsymbol{q},\ \dot{\boldsymbol{q}}) = \boldsymbol{\tau}, \qquad (3)$$
$$\boldsymbol{F}(\boldsymbol{q},\ \dot{\boldsymbol{q}}) := \boldsymbol{C}(\boldsymbol{q},\ \dot{\boldsymbol{q}}) + \boldsymbol{V}\dot{\boldsymbol{q}} + \boldsymbol{G}(\boldsymbol{q}).$$

In order to construct the simulation model, i.e., the time derivative of the state vector

$$\frac{d}{dt}\begin{bmatrix}\boldsymbol{q}\\ \dot{\boldsymbol{q}}\end{bmatrix} = \begin{bmatrix}\dot{\boldsymbol{q}}\\ \boldsymbol{M}(\boldsymbol{q})^{-1}(-\boldsymbol{F}(\boldsymbol{q},\ \dot{\boldsymbol{q}}) + \boldsymbol{\tau})\end{bmatrix}, \qquad (4)$$

$\boldsymbol{M}(\cdot)$ and $\boldsymbol{F}(\cdot)$ are symbolically extracted from (3). By evaluating them using identified base parameters, the right hand side of (4) becomes fixed function vector of $\boldsymbol{q}$ and $\dot{\boldsymbol{q}}$, which serves as the simulation model.
The regression model for parameter identification based on equations of motion can be similarly constructed as (1):

$$\phi_d(\boldsymbol{q}, \dot{\boldsymbol{q}}, \ddot{\boldsymbol{q}})^T \boldsymbol{a} = \boldsymbol{\tau} \qquad (5)$$

by extracting the coefficients of the base parameters from (3).
Model reduction/transformation: Obtaining a model for designing controllers may involve further symbolic manipulation of (3). To design a standard linear controller for example, we need a linearly approximated model of (4) around given operating point $[\boldsymbol{q}^{oT}, \dot{\boldsymbol{q}}^{oT}]^T$ in the state space. To this end, series expansion of (4) is symbolically computed to give

$$\frac{d}{dt}\begin{bmatrix}\tilde{\boldsymbol{q}}\\ \dot{\tilde{\boldsymbol{q}}}\end{bmatrix} = \boldsymbol{A}\begin{bmatrix}\tilde{\boldsymbol{q}}\\ \dot{\tilde{\boldsymbol{q}}}\end{bmatrix} + \boldsymbol{B}\boldsymbol{\tau}, \qquad (6)$$

where $\tilde{\boldsymbol{q}} := \boldsymbol{q} - \boldsymbol{q}^o$. (6) will be exported as two constant matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ for further numerical computations. Other transformations are also possible; for example, the feedback linearization are computed using Lie algebra, which is accomplished essentially by symbolic differentiations and symbolic matrix manipulations.

Figure 1 shows the flow of the modeling process. The structural modeling is performed by symbolic operations, and it terminates by separating symbols in two kinds: parameters and variables. Once the symbols are separated, further processing can be handled by numerical tools.

The modeling procedure similar to the above are implemented in several modeling program packages, such as *Robotica* (on *Mathematica*), ROSAM (on Maple), SYMORO+ (on *Mathematica*; see [3]). The authors also developed a program package *Moderato* on *Mathematica* to execute the symbolic modeling process. There also exist program packages for numerical modeling such as Robotics Toolbox (on Matlab) and SpaceDyn (on Matlab).

3 Controller design and the simulation

3.1 Use of a numerical tool

Numerical simulations require numerical evaluation of equations of fixed form. The use of simple data format for the variables, such as the standard floating point number, e.g., is preferable for the fast evaluation. Therefore, the fixed objects generated by symbolic operations will be transformed into numerical objects and exported into the numerical computation environment. When exporting objects, how easily it is done is the most important point.

In some cases, however, thus exporting objects may not be necessary. For example, *Mathematica* can handle objects of simple data format, generated by compiling symbolic object with `Compile[]` function. Since evaluation of the compiled object is performed faster, fast numerical operations is also possible on *Mathematica*; thus the aim is achieved. Still, since most of the current controller design algorithms are finally translated into numerical operations, and there are a lot of numerical CAD software on which these algorithms are implemented, converting into numerical objects is currently practical, and convenient for further computations.

3.2 Exporting objects into MATX environment

The authors used MATX CAD software package [1] as the numerical tool. MATX is a programming language with C like syntax and has many sophisticated data types for control system design. It provides two kinds of processing styles: an interactive session environment `matx` and a compiler `matc`. The latter compiles MATX source codes into the 'native' objects and generates a executable code on the operating system by linking the objects. Moreover, `matc` can handle (link) other native objects no matter which compiler generates them; it may be C, FORTRAN, or other compilers. Many existing resources written in these languages can be directly incorporated.

In spite that *Mathematica* has C code generating function, `CForm[]`, the authors developed a similar function `MaTXForm[]` on *Mathematica* which translate *Mathematica* object into MATX syntax to export symbolic objects as source code of MATX (**List 1**). This maximally utilizes the syntactic advantage of MATX over C language while `CForm[]` approach is also possible.

```
In[20]:= H

Out[20]= {{a1 + 2 a3 Cos[q2[t]], a2 + a3 Cos[q2[t]]},

>    {a2 + a3 Cos[q2[t]], a2}}

In[21]:= MaTXForm[H]

Out[21]= [[a1 + 2*a3*Cos(x(2)), a2 + a3*Cos(x(2))][a2\

>    + a3*Cos(x(2)), a2]]
```

List 1: Translation by `MaTXForm[]` (on *Mathematica*)

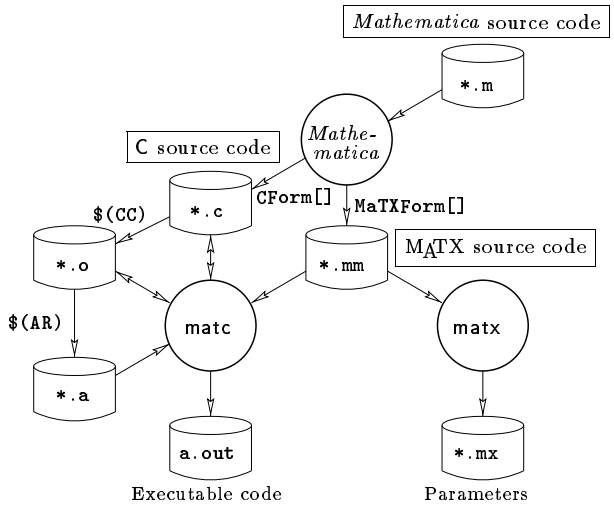

Figure 2: Combination of MATX and *Mathematica*

The simulation model, i.e., $\boldsymbol{M}(\cdot)$ and $\boldsymbol{F}(\cdot)$ in (4) are exported (saved in a file) and merged into the source code of MATX; it will be processed by `matc` to generate a binary object code. On the other hand, the controller design model, $\boldsymbol{A}$ and $\boldsymbol{B}$ in (6) are included into a source code for the controller design; it will be processed by `matx`.

Figure 2 shows the relations between files in the combined *Mathematica*/MATX environment described above. The code for symbolic modeling (`*.m`) is processed by *Mathematica*, and the computed models are saved as MATX source code (`*.mm`) or in C format (`*.c`). The simulation model, if it is self-contained, is compiled with `matc` to generate a executable file (`a.out`). Otherwise, it is compiled into an object module (with unresolved symbols; `*.o`), and finally linked together with other object modules into a executable file; `matc` works as the linker. This separated compilation matches to the different update-frequencies of the modules; usually, the simulation model is updated less frequently than the controller. The simulation model and the relating materials are archived into a file as a library of object modules (`*.a`); this makes a way of maintaining a database of models in suppressed form. The model for designing controllers is incorporated into a controller design program. It is interactively processed with `matx`. The resulting controller is saved as constant matrices in a file (`*.mx`) and will be used in the closed-loop simulation.

4 Example: The Spherical Inverted Pendulum

This section describes the modeling and the linear controller design using *Mathematica* and MATX for stabilizing the spherical pendulum.

The pendulum system consists of a solid aluminum rod

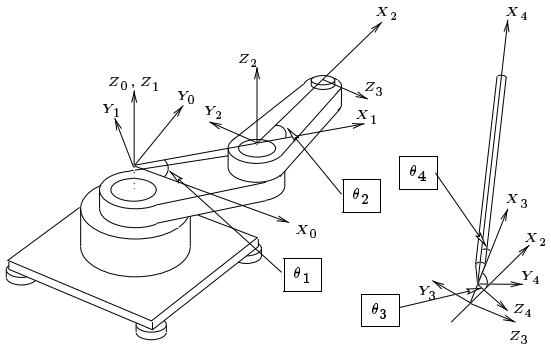

Figure 3: The spherical inverted pendulum

Table 1: Modified D-H parameters

i	a_i	d_i	α_i	θ_i
1	0	0	0	θ_1
2	a_2	0	0	θ_2
3	a_3	0	$\pi/2$	θ_3
4	ε (=0)	0	$\pi/2$	θ_4

Table 2: Inertial parameters

i	J_{xxi}	J_{yyi}	J_{zzi}	s_{xi}	s_{yi}	s_{zi}	m_i
1	J_{xx1}	J_{yy1}	J_{zz1}	l_1	0	s_{z1}	m_1
2	J_{xx2}	J_{yy2}	J_{zz2}	l_2	0	s_{z2}	m_2
3	0	0	0	0	0	0	0
4	$\simeq 0$	$\rho l^3/12$	$\rho l^3/12$	$l/2$	0	0	ρl

and a SCARA manipulator (Figure3), and can be regarded as a serial connection of four rigid links. The third link is virtual. Four local coordinates are attached as Figure3. The corresponding modified D-H parameters are determined as Table 1, which completes the kinematics modeling. The inertial parameters are measured in each coordinate as Table 2; since each body is symmetric, the off-diagonal entries of inertial tensors are zero. ρ and l are the linear density and the length of the pendulum rod.

List 2 and List 3 are the parameter files for *Mathematica*. By grouping the inertial parameters, 8 parameters are reconstructed as the base inertial parameters as

```
(* gravity acceleration *)
  g = {0, 0, -G};

(* D-H parameters *)
  a = {0, a2, a3, 0};
  d = {0, 0, 0, 0};
  Alpha = {0, 0, Pi/2, Pi/2};
  q = {q1[t], q2[t], q3[t], q4[t]};

(* generalized coordinate *)
  gc = q;

(* operating point *)
  op = {0, Pi/2, Pi/2, 0, 0, 0, 0, 0};

(* actuated joints *)
  AJ = {1, 2};

(* measurements *)
  Mes = q;
```

List 2: Parameters for symbolic modeling

```
dynp = {
  J1xy -> 0, J1xz -> 0, J1yz -> 0,
  J2xy -> 0, J2xz -> 0, J2yz -> 0,
  J3xx -> 0, J3xy -> 0, J3xz -> 0,
  J3yy -> 0, J3yz -> 0, J3zz -> 0,
  J4xx -> 0, J4xy -> 0, J4xz -> 0,

  J4yy -> rhol*l^3/12,
  J4yz -> 0,
  J4zz -> rhol*l^3/12,

  s1x -> l1,  s1y -> 0,
  s2x -> l2,  s2y -> 0,
  s3x -> 0,   s3y -> 0,  s3z -> 0,
  s4x -> l/2, s4y -> 0,  s4z -> 0,

  m3 -> 0,  m4 -> rhol*l,
  V3 -> 0,  V4 -> 0
}
```

List 3: Description of dynamics parameters

shown in Table 3; the relation to the original parameters are given as

$$\begin{aligned} J'_{xx3} &= \rho l^3/3 \\ J'_{xx4} &= -\rho l^3/3 \\ J'_{zz1} &= J_{zz1} + m_1 l_1^2 + m_2 a_2^2 + \rho l a_2^2 \\ J'_{zz2} &= J_{zz2} + m_2 l_2^2 + \rho l a_3^2 \\ J'_{zz3} &= \rho l^3/3 \\ J'_{zz4} &= \rho l^3/3 \\ ms'_{x2} &= m_2 l_2 + \rho l a_3 \\ ms'_{x4} &= \rho l^2/2. \end{aligned} \tag{7}$$

Further symbolic processing gives two models: the simulation model, i.e., $\boldsymbol{M}(\cdot)$ and $\boldsymbol{F}(\cdot)$ in (4), saved in `math/M.mmx` and `math/F.mmx` and the linearly approximated model, i.e., $\boldsymbol{A}$ and $\boldsymbol{B}$ in (6), saved in `math/A.mmx` and `math/B.mmx`. Thus the symbolic modeling is completed.

The simulator is defined as a function `pend()` (List 4); the expressions of two matrices `M` and `F` are read from the files generated by *Mathematica*. The time derivative of the state vector is numerically computed. The parameter values are defined in the header files. List 5 is the controller design code. It reads two matrices `A` and `B` from the files; these are also generated by *Mathematica*. The linearly approximated model (`A`, `B`) is transformed into the discrete-time model (`Phi`, `Gamma`). The discrete-time LQ optimal controller is computed, and the feedback gain `F` is saved in a file `ctrl.mx`.

The simulation is carried out by executing `sim`, reading `ctrl.mx`, and the result is saved in a file `sim-log`. `sim` is generated by linking the main body (`sim.mm`) and the simulator (`eqn.mm`) through a library (`libpend.a`).

Table 3: Base inertial parameters

i	J'_{xxi}	J'_{yyi}	J'_{zzi}	ms'_{xi}	ms'_{yi}	ms'_{zi}	m'_i
1	0	0	J'_{zz1}	0	0	0	0
2	0	0	J'_{zz2}	ms'_{x2}	0	0	0
3	J'_{xx3}	0	J'_{zz3}	0	0	0	0
4	J'_{xx4}	0	J'_{zz4}	ms'_{x4}	0	0	0

```
#include "mdefs.h"
#include "scara.h"
#include "pend.h"
#include "base.h"

Func void pend(dx, t, x, u)
  Real t;
  Matrix x, dx, u;
{
  Matrix M, F;

  M =
#include "math/M.mmx"
;
  F = trans(
#include "math/F.mmx"
);

  dx = [[x(5:8)][M~*(-F + u)]];
}
```

List 4: Simulation model (MATX: `eqn.mm`)

```
#include "mdefs.h"
#include "scara.h"
#include "pend.h"
#include "base.h"

Func void main()
{
  Matrix A, B, Q, R, P, F, Phi, Gamma;
  Real dt;

  Q = diag(12000.0, 3000.0, 5000.0, 1000.0,
           800.0, 500.0, 500.0, 300.0);
  R = diag(1.0, 1.0);
  dt = 0.0084;

  A =
#include "math/A.mmx"
;
  B =
#include "math/B.mmx"
;

  {Phi, Gamma} = c2d(A, B, dt);
  P = DRiccati(Phi, Gamma, Q, R);
  F = - (R + Gamma'*P*Gamma)~*Gamma'*P*Phi;

  print F, dt -> "ctrl.mx";
}
```

List 5: Controller design code (MATX: `lin.mm`)

```
Matrix F, op;
void pend();

Func void main()
{
  Real dt;
  Matrix x0;
  Array TH, XH, UH;
  void lq();

  read F, dt <- "ctrl.mx";

  op = [[0.0, PI/2, PI/2, 0.0], Z(1, 4)]';
  x0 = op + [[0.1, 0.1, 0.0, 0.0], Z(1, 4)]';

  {TH, XH, UH} = Ode45HybridAuto(0.0, 5.0,
                 dt, x0, pend, lq);

  print [[TH][XH][UH]] >> "sim-log";
}

Func void lq(u, t, x)
  Real t;
  Matrix u, x;
{
  u = [[F*(x - op)][Z(2, 1)]];
}
```

List 6: Simulation code (MATX: `sim.mm`)

```
SRCS = sim.mm
LIBSRCS = eqn.mm mes.mm
HDRS = scara.h pend.h base.h mdefs.h

MATXCC = matc
MATXLD = matc
AR     = ar rcv
RANLIB = ranlib
CFLAGS = -O2
LDFLAGS = -L.

OBJS = $(SRCS:.mm=.o)
LIBOBJS = $(LIBSRCS:.mm=.o)

.SUFFIXES: .mm

.mm.o:
        $(MATXCC) $(CFLAGS) -c $<

sim-log:        sim ctrl.mx
        sim

sim:    $(OBJS) libpend.a
        $(MATXLD) $(LDFLAGS) -o sim $(OBJS) -lpend

libpend.a:      $(LIBOBJS)
        $(AR) libpend.a $(LIBOBJS)
        $(RANLIB) libpend.a

eqn.o:  $(HDRS) math/M.mmx math/F.mmx
mes.o:  $(HDRS) math/mes.mmx

ctrl.mx:        lin.mm $(HDRS) math/A.mmx math/B.mmx
        matx lin.mm -e 'main();'
```

List 7: `Makefile` for simulation

List 7 is a `Makefile` for **make** utility; it shows the dependencies of files and the relating operations which have been described. **matc** works just like as a usual compiler.

Since the simulation result can be saved as a simple text file, several kind of post processing for the visualization are possible; Figures 4 and 5 are the examples.

In this spherical pendulum case, given the kinematics and the dynamics parameters, the symbolic modeling was accomplished in 20 *seconds*, and the numerical simulation took less than 20 *seconds* including the code compilation time, on a PC with PentiumII 400 [MHz] running Linux. Thus the described method and the tools make the modeling and the simulation task easy.

5 Concluding remarks

The combinational use of symbolic and numerical tools in modeling mechanical systems and in performing the simulations has been described. An example of the spherical pendulum illustrates the concrete modeling procedure. This mixed usage approach is considered to be effective other than mechanical systems when the modeling consists of two phases: the structural modeling and the quantitative modeling.

Acknowledgement: The first author would like to express deep gratitude to Dr. Masanobu Koga, who developed and is maintaining MATX, for his valuable comments in preparing this paper.

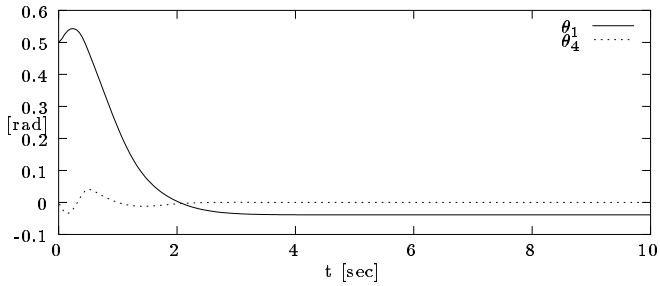

Figure 4: Simulation result (θ_1 and θ_4)

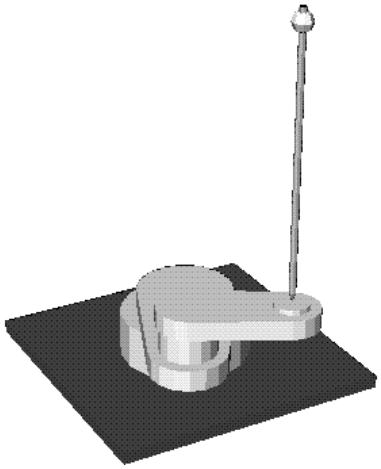

Figure 5: 3-D visualization

References

[1] M. Koga and K. Furuta, "A high-performance programming language (interpreter and compiler) for scientific and engineering computations," *in Proc. IEEE Int. Symp. on Comp.-Aided Contr. Sys. Design,* Napa, pp. 15–22, 1992.

[2] S. L. Campbell, F. Delebecque and D. von Wissel, "A mixed symbolic-numeric software environment," *in Proc. IEEE Int. Symp. on Comp.-Aided Contr. Sys. Design,* Dearborn, pp. 436–441, 1996.

[3] W. Khalil and D. Creusot, "SYMORO+: A system for the symbolic modelling of robots," *Robotica,* vol. 15, pp. 153–161, 1997.

[4] T. Hoshino and K. Furuta, "Stabilization of 2-D inverted pendulum via partial exact linearization," *in Proc. Asian Contr. Conf.,* vol. 2, pp. 495–498, 1997.

[5] C. C. de Wit, B. Siciliano, and G. Bastin (Eds), *Theory of Robot Control,* chapter 1, Springer-Verlag, 1996.