Astronomical Data Analysis Software and Systems IV
ASP Conference Series, Vol. 77, 1995
R. A. Shaw, H. E. Payne, and J. J. E. Hayes, eds.

The Stellar Dynamics Toolbox NEMO

P. Teuben

University of Maryland, College Park, MD 20742

Abstract. NEMO is a toolbox for users and programmers with which a wide range of experimental situations can be constructed, dynamically evolved, and analyzed. Primarily geared towards stellar dynamics, NEMO is by no means restricted to it. Great importance has been given to making the package extendible and to flexibly import and export data. NEMO has been implemented within, and integrated into, the UNIX environment. A unified command syntax provides users with a simple help facility, and provide programmers with a ready-made user interface. A general method for hierarchically structured binary data files is used to communicate between programs. We also describe a proposal to adopt FITS BINTABLEs as a basis for interchanging and archiving simulation results.

1. Introduction

With the advent of powerful workstations in the mid 1980s, the use of N-body simulations has become increasingly widespread. It was realized (e.g., Hut & Sussman 1986; Hut 1989) that this part of the astronomical community would benefit from a programming environment similar to what many observers have had for years in packages like AIPS and IRAF. Most researchers in the field have their own private packages—some of them with a large degree of sophistication. A first attempt to structure these efforts into a package was reported by Barnes et al. (1988), and became known as NEMO. In this paper we report on the current status of NEMO (see also Teuben 1994) and its future prospects.

2. System Components

To the user, NEMO appears as a collection of loosely coupled programs. The following components can be identified within NEMO:

A common user interface is shared by all programs; all programs are executed from the UNIX shell and accept a list of *keyword=value* pairs on the command line. There are **program** and **system** parameters, the latter shared by all programs. These system parameters control global properties such as error handling, debug output, graphics devices, etc. The user interface allows you to switch to a different look-and-feel user interface. Examples have been constructed in interfacing NEMO with the Miriad (an AIPS-like interface, see Sault et al. 1995) and KHOROS/CANTATA (a visual programming language) packages. In practice, however, we find shell scripts ("batch mode") the most powerful and self-documenting mode of operation.

A common file-structure is shared by all programs. Data, used to communicate between programs, is stored in files in a hierarchical structure of name- and type-tagged items, in binary format. Items can be single scalars, multi-dimensional arrays, and even arbitrary (C-type) structures. In addition, each program can have an input and output channel which is connected to a UNIX pipe instead of a disk-file. Since many programs transform one dataset into another, this allows for efficient chaining to build new tools.

On top of this file-structure, a number of packages have been defined. The most important one is *snapshot*, on which NEMO was originally based. Subsequently *images*, *tables*, and *orbits* were introduced, with a number of programs that convert data between them.

A common graphics interface is defined on the Applications Programming Interface (API) level. One simply needs to link with any graphics library (e.g., MONGO, PGPLOT, X11, PostScript, SunView, or GL) to create a working program. This hides the complexity of different graphics programs but is, of course, not as flexible since a fairly low common denominator has to be chosen for the API.

Programs can use a dynamic loader in order to compile, load, and run user-specific code. This supports very flexible and powerful analysis tools. For example, in orbit integration, a user-defined potential can be used to integrate and analyze orbits. And for snapshots, arbitrary user-defined expressions are used for many analysis and plotting programs.

3. Documentation

Extensive on-line help is present on three levels. First of all, each program comes with a dynamic inline help, which can remind the user of the keywords, their default values, and a small one line description of each keyword. It is part of the user interface for which the programmer is responsible. The advantage of this lowest help level is that it is much less likely to be out of date than the other levels, and does not necessarily need the entire NEMO environment to run.

The second form of help is a standard UNIX manual page created for each program. This is an example of how NEMO is integrated into the UNIX environment and where "standard" tools like `xman` and `tkman` can be immediately used for powerful (hypertext) browsing.

The third form of help is the extensive "Users and Programmers Guide." It contains numerous examples of how to use, program, and extend the toolbox. NEMO information is also available on the World Wide Web[1].

4. Future

After the on-going conversion of supporting Solaris 2.x, NEMO will have formal ANSI C and C++ support. Also, support for other packages, such as Star-

[1] http://astro.umd.edu/nemo

lab (Hut, Makino, & McMillan, private communication), and TIPSY (Katz & Quinn, private communication) is on-going.

5. Example

To illustrates some of the features in NEMO, the following C-shell script sets up an encounter between two 256 body Plummer models, evolves the encounter, and plots the evolution in a 3 by 3 panel:

```
mkplummer out=mod1 nbody=256
mkplummer mod2 1024/4
snapstack in1=mod1 in2=mod2 out=mod5 deltar=10,2,0 deltav=-0.6,0,0

hackcode1 in=mod5 out=mod5.out freqout=1 tstop=80  > mod5.log

snaptrim in=mod5.out out=- times=80 |\
    hackforce - - |\
    snapcenter - - 'weight=-phi*phi*phi' report=f |\
    unbind - - |\
    snapplot - xrange=0:9 yrange=-3:3 nxticks=5 nyticks=5 \
        xvar=r 'yvar=(x*vy-y*vx)/sqrt(x*x+y*y)' \
        times=0,10,20,30,40,50,60,70,80 nxy=3,3
```

Note that command line parameters do not have to be specified by name but can also be passed by order. UNIX pipes can be used to chain commands, in which case the files which denote the ends of the pipe must be denoted with a "-" symbol. Lastly, numeric values can generally be given as arbitrary expressions.

6. N-Body Data Interchange Format

Over time, NEMO has accumulated a number of routines that support import from, and export to, a large variety of ASCII and binary formats. A draft was written (Teuben 1994) where the FITS BINTABLE (Cotton, Tody, & Pence 1995) has been used to define a conceptual format with which different packages can exchange their simulation results. The FITS BINTABLE extension is well documented, and allows for efficient storage of binary tables. Each row represents a particle, and different columns represent different attributes of the particle (a column can be a vector), such as mass, position, velocity, potential, density, etc.

The BINTABLE format is merely a prescription for storing tabular data. Any FITS reader should be able to read (and/or skip) this NBODY data, but only specialized readers would know what to do with this data. There are a number of implementation details that still have to be decided. We note the following:

- A naming convention for the column names (the so-called `TTYPE`'s) is required. This will be the most visible part of the implementation. Apart from registering a new extension name (e.g., NBODY) with the FITS standards office, the community needs to agree on things like the column names, units, coordinate systems, etc.

- Datasets with different kinds of particles which have different attributes. For example, SPH particles have extra attributes (such as temperature and density) which pure N-body particles formally don't have. It would not be very efficient to tag along such unneeded data attributes. The "variable length array" construct within the BINTABLE proposal could solve this problem, but this has to be worked out in more detail. A more obvious solution is to store them in separate FITS extensions. This is clearly a much simpler and easier to understand implementation, but requires FITS readers to look ahead.

- Some variables are global variables. There may be more efficient ways to store these values. For example, if all the masses (`MASS`) are the same, they could be stored in the header (which is, however, in ASCII). Another example is `TIME`. Although, in principle, each particle may have it's own variable time, a snapshot has, by definition, a single time shared by all particles. The format needs to allow for both.

- For certain complex situations, extra data structures may be needed to fully describe an experimental situation. For example, descriptions of hierarchical double and triple stars could be stored in an accompanying table. The names and descriptions of these extensions would need to be part of the standard, too, but they can always be added to the proposal at a later date. This is a very common technique in radio interferometry (VLBA data is stored with many associated tables).

Acknowledgments. NEMO was initially developed by Joshua Barnes, Piet Hut, and myself during the 1986–87 academic year in Princeton. I wish to thank them for the stimulating environment in those years.

References

Barnes, J. E., Hernquist, L., Hut, P., & Teuben, P. 1988, BAAS, 20, 706

Cotton, W. B., Tody, D., & Pence, W. 1995, Binary Table Extensions to FITS, in preparation

Hut, P. 1989, Celestial Mechanics, 45, 213

Hut, P., & Sussman, G. J. 1986, in The Use of Supercomputers in Stellar Dynamics, ed. P. Hut & S. McMillan (Berlin, Springer-Verlag), p. 193

Sault, R. J., Teuben, P. J., & Wright, M. C. H. 1995, this volume, p. **??**

Teuben, P. 1994, PASJ, in press