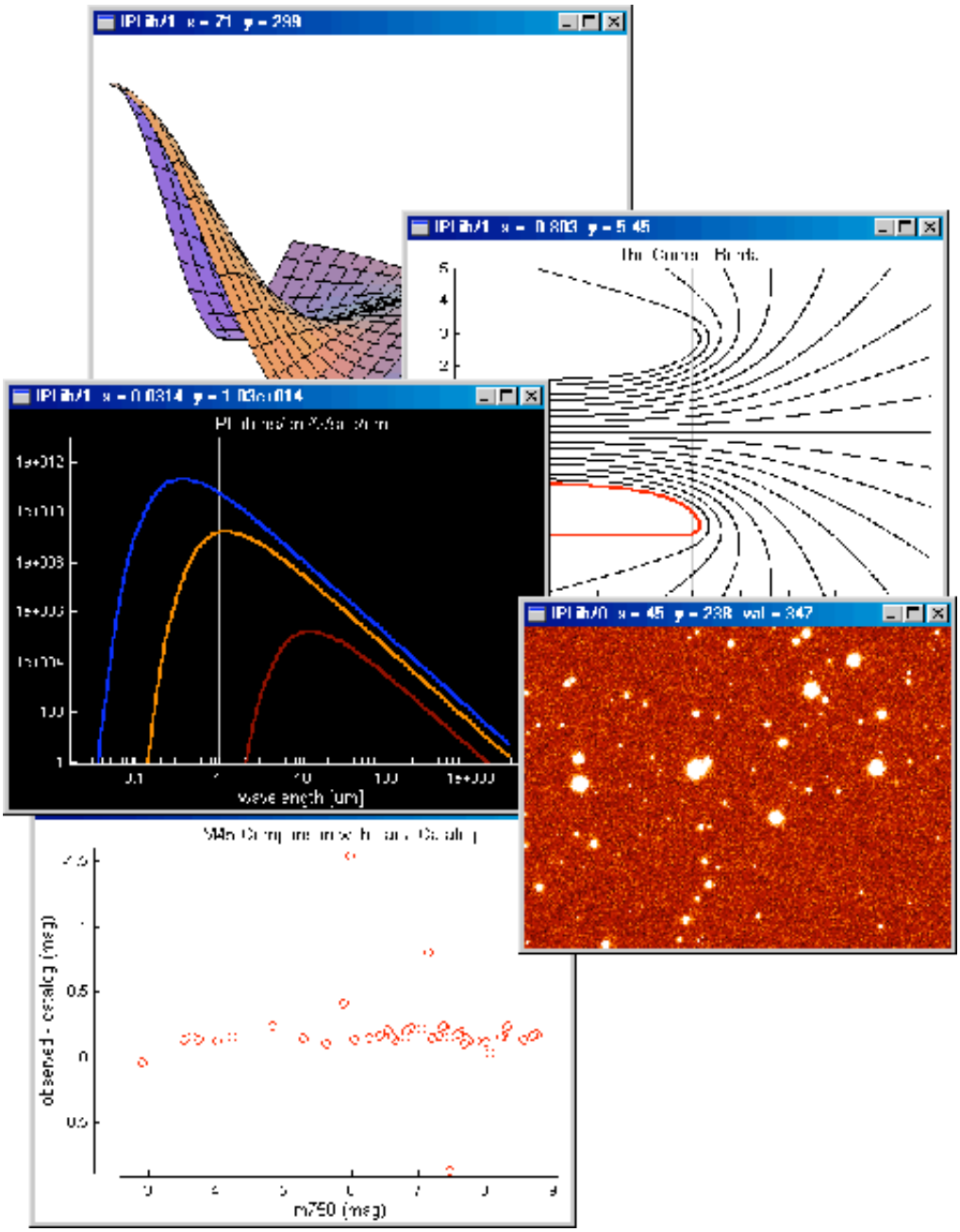
NML – A Tool for Numerical Modeling and Data Analysis

D.McClain, Sr. Scientist, Raytheon Systems Co., Tucson, AZ

NML is officially shareware within the restrictions imposed by the OCaml license from INRIA, France.

Modifications to the source are freely permitted with the request that we receive official acknowledgment within the source files, and that you take responsibility for your modifications.



NML is a dynamically typed functional language whose syntax conforms closely to that of OCaml. In addition, it supports overloaded, vectorized, math operations, list comprehensions, and optional and keyword arguments in uncurried argument tuples, possibly with specified default values. It can access OLE compliant, and low-level COM interfaces, supports serial I/O and socket based communication, and provides an ADO connection to external databases. An Emacs mode is supported through a hacked Tuareg interface, as well as a Tcl/Tk interactive browser and interaction window with list pane access to a user modifiable documentation database.

NML compiles its source code to fast closures of native OCaml code at speed of about 10K lines/sec. It features autoloading of dependent modules. Each module (1 module = 1 source file) defines its own namespace. Access to bindings defined in other modules

is possible through dotted long-idents or by declaring a module "open". Module abbreviations, "with_module" ,and user definable types are supported. Pattern matching reflects the syntax of OCaml with the addition of basic type matching patterns, and Prolog list syntax. Data types include int, float (64-bit), complex, symbol, string, char, list, vector, tuple, numeric multidimensional arrays (up to 2^31 elements), I/O channels, Stacks, Mappings, Hashtables, Buffers, Regexps, and more…

NML handles the math stuff at speeds twice that of RSI/IDL modeling language, but even more it handles the non-numeric stuff painlessly with its functional approach, and the huge supporting library inherited from OCaml. Numerous sample modules are provided to help you. In fact one of the samples reimplements Norvig's Prolog interpreter in fewer lines than Lisp. Numeric modeling has been available from a number of proprietary sources, but one finds that after the math stuff is so quickly dispatched, the remaining non-numeric stuff is a real pain… Not so with NML. It readily handles both numeric and non-numeric algorithms with equal brevity.

Documentation is currently quite brief. The sources are the ultimate documentation, of course, but there is an Access'97 database included with thumbnails of many of the routines. This database can be browsed using a provided Tcl/Tk based NML interactor, by using Access'97, or by using the prefix "??" operator on a module name and the name of the binding of interest within that module. You should arm yourself with the OCaml documentation, especially for use with the standard libraries, most of which are made part of NML. Functions "abbrev" and "apropos" also exists, along with "vlist" for listing the vocabularies of modules.

Functions can define uncurried tuples with both optional and keyword arguments using the Lisp notation "&optional", "&key", and "&rest". Complex numbers can be entered directly using a syntax like "1.2+3.4i". Aggregate types, tuple, list, vector, and numeric array, can be readily converted from one form to another. Array access proceeds by using the ".[…]" syntax of OCaml, but for multidimensional arrays, slices can be defined in any dimension. For example, "x.[1:10, *, 3]" demonstrates one way of extracting element 3 of the row-major dimension, all rows, and aggregates of rows numbered 1 through 10. Array indexing begins at zero. Negative array indices and indices beyond array bounds are handled by wrapping them modulo their corresponding dimension. This is the most useful mapping of indices when dealing with Fourier techniques for image processing. Hence, x.[-1] refers to the last element of array x. Arrays exist simultaneously in both multidimensional form and a 1 dimensional vectors of unboxed doubles. An NML vector type also exists, but this is identical to an OCaml array. Conversion from list, and vector to numeric array is automatic for most math operations.

Unlike OCaml, lists, and vectors can contain any mixture of data types. Tuples, lists, vectors, strings, arrays, and even scalar values can all be accessed using the simple array indexing demonstrated above. Since there is no compile time checking or inference of data types, it is possible to create unsafe programs, in the sense that a runtime error can occur through misuse of routines. However, all bindings are checked at compile time to ensure their existence before use. That catches the common spelling errors. Unused patterns in matchings are indicated with a warning, but patterns can contain any mixture of data types as well. NML is fully tail pure for both recursion and function calls from tail position. An infinite loop that generates data will run forever without blowing the stack. Exception handling is provided, as in OCaml, using "try … with". In addition to pattern matching it also offers a "case" statement which is shorthand for a series of nested "if then else" clauses.

NML/IPLib can be used in a variety of ways:

1. Direct OCaml bindings permit its use as an add-in library
2. Standalone as an interactive programming environment.
3. As a DLL to be called from any language with a C FFI
4. As a low-level COM library for language independent interfacing
5. As an IDispatch library for calling from VBasic and other languages
6. Interfaces exist for C/C++, VBasic, OCaml, Dylan, Mathematica, Lisp, and Scheme.

There is much, much, more… If you are interested, you can obtain the complete system with source code at http://www.azstarnet.com/~dmcclain/nml.zip (1100 KB). Contact info, dmcclain@azstarnet.com.