REC-R for Second Order Real Differential Equations

Harold V. McIntosh
Departamento de Aplicación de Microcomputadoras,
Instituto de Ciencias, Universidad Autónoma de Puebla,
Apartado postal 461, 72000 Puebla, Puebla, México.

January 8, 2001

Abstract

REC-R is one of a series of specialized REC programs; in this case one which provides assistance in graphing the solutions of real, second order differential equations. It is therefore an adjunct to the program `SERO` which serves the same purpose.

1 Introduction

Three ingredients are required to understand this program. Since it destined for the specific purpose of displaying the solutions of differential equations, familiarity with that subject matter is obviously required. Additionaly, it is intended to be a user interface appendage to the program `SERO`, implying a familiarity with that program as well.

Given that the design is accomplished by using REC, an understanding of that language is evidently required.

Finally, the program is implemented in Objective C running under the NeXTSTEP operating system [3], to take advantage of its excellent visual interface and other attractive features.

1.1 Second order real differential equations

As is typical of all branches of analysis, differential equations should be studied in the complex domain. However, programming languages duch as `FORTRAN` or `C` have traditionally encountered problems when being asked to deal with complex numbers and even if that were not so, the increased running time and memory requirements work against their use. Fortunately, a large part of the results which one requires can already be obtained in the real domain.

There are numerous books on differential equation theory. One which treats spectral theory in more detail than most is Coddington and Levinson [5]. Another, which has good material relevant to second order equations in the phase plane is Hartman [6].

SERO is based on the reduction of a second order linear differential equation to a pair of first order equations written in matrix form, a format from which it can evidently handle any other pair of first order equations, such as a one dimensional Dirac equation, whatever their origin. In other words, it considers matrix differential equations of the form

$$\frac{d\mathbf{Z}(t)}{dt} = M\mathbf{Z}(t) \tag{1}$$

where M is a 2×2 matrix of coefficients, some or all of which are functions of the independent variable.

SERO groups these elements into a "potential matrix" which is predefined in the program, of which one of several is selectable by a REC operator. No use has been made of a run-time compiler, even one such as YACC, so the coefficient matrix must be foreseen and provided for before running the program.

All numerical integration is carried out by either a fourth order Runge-Kutta program, or a sixth order program. It is a general custom for programs to restrict themselves to the fourth; for one thing, it is counterproductive to choose a step size greater than the graphing interval (otherwise supplementary interpolation is required), so there is no point to using high order methods to advance the solution point rapidly. For another, the fourth order method is a good break-even point for increased accuracy versus increased complexity of computation.

1.2 The programming language REC

The programming language REC was derived from LISP over a period of time, and found to be useful for minicomputers such as Digital Equipment's PDP-8, or early microprocessors such as the IMSAI or Polymorphic 88, given its extreme conciseness. That, in fact, is the reason why it is still preserved as an interface to programs written in other languages running with other operating systems.

REC itself is strictly a control structure, using parentheses to group sequences of instructions which are supposed to be executed in order — in other words, programs. The instructions themselves, which are subdivided into operators and predicates, are not part of the control structure, but are always defined separately to create specialized variants on REC.

Besides the use of parentheses for aggrupation, the two punctuation marks, colon and semicolon, are used for repetition and termination, respectively; they were somewhat borrowed from musical notation. Spaces, tabs, all the `C` "white space," are used for cosmetic purposes and must be ignored.

There is also a mechanism for defining subroutines; definitions and their associated symbols alternate between a balanced pair of curly brackets, terminating in a main program which remains nameless. Definitions so introduced are valid only within their braces, permitting symbols to be reused over and over again on different levels and in different places. A subroutine is actually executed by prefacing its name with an "at sign," as in `@x`.

The distinction between operators and predicates is somewhat artificial, made mostly for convenience; a predicate which is always `true` is an operator. The values `true` and `false` of a predicate govern the sequence of operations, `true` meaning that the text of the program continues to be read in order, consecutively from left to right.

But `false` makes use of the punctuation, implying a skip to the next colon, semicolon, (or right parenthesis, in their absence) at the same parenthesis level. A colon means repetition starting back at the left parenthesis, a semicolon gives a `true` termination, realized by going forward to the closing right parenthesis, Arriving at a right parenthesis without the benefit of punctuation gives a `false` termination, but more generally inverts the action of predicates enclosed in such an interval.

This convention allows negating a predicate: (p) behaves oppositely from p. The other boolean combinations of predicates are easily written; “p and q” translates into (pq;), “p or q” into (p;q;). Moreover, all programs are predicates, even the main program, thereby reporting their results to the next higher program level, the one in which they occur as a subroutine.

1.3 the specific vocabulary of REC-R

Having a specialized REC for solving differential equatioms, or more precisely, graphing their solutions, we need to survey its vocabulary. Fourth, or possibly sixth, order Runge-Kutta is the central process, advancing from one grid point to the next. Paralleling this movement, any of the elements of the coefficient matrix or of the solution matrix (or its trace, useful for stability arguments) can be made to follow along with cursor-moving operations. By maintaining horizons and providing for offsets, the graphing can be done with hidden line suppression, albeit in its most primitive form without exact intersections of the curves with their horizons.

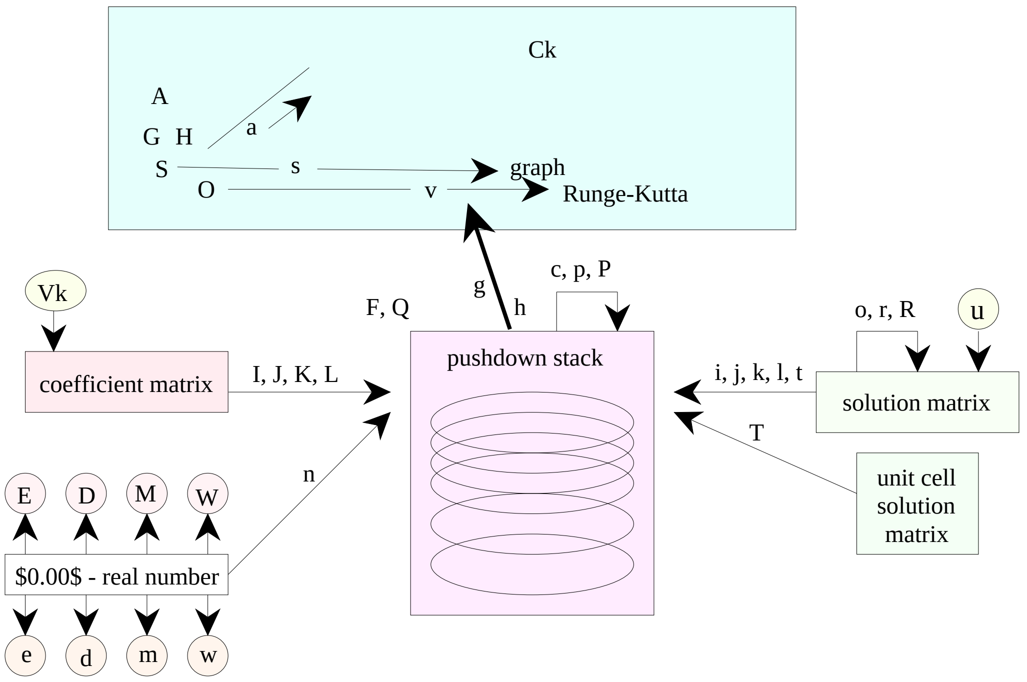


Figure 1: Data movements caused by the REC operators.

REC-R assigns practically the whole alphabet to operators and predicates. However, they can be grouped into just a few categories, which can now be mentioned, although fuller descriptions will be given later on in another section. Basically, there is a very short pushdown list, which mostly holds the coordinates of a graph under construction.

Two graphing styles are available, both with possible hidden-line suppression. One represents elements of the Runge-Kutta matrices as functions of the independent variable, the other graphs some components as functions of others, as befits a phase plane. The choice depends on the selection of coordinates for the line drawer.

A variety of parameters can be defined by using the floating point number operator and then sending its value to a destination, which could even be the top of the pushdown stack.

Finally there are the Runge-Kutta integrators, the pen movements, and some predicates which can be used to vary pen parameters.

1.3.1 generic REC symbols

[...] - enclose a comment

{ ... } - enclose a definition list

(...) - enclose a REC expression

;, : - exit at expression's end, repeat from beginning

@x - call the subroutine x

0.00 - create a real number

!n! - (!n! xxx :;) repeat xxx n times

1.3.2 matrix elements and loading the pushdown stack

Referring to Equation 1 to identify the matrix elements,

i, j, k, l - push elements $z_{11}, z_{12}, z_{21}, z_{22}$

I, J, K, L - push elements $m_{11}, m_{12}, m_{21}, m_{22}$

B, b - check sign, confinement of Trace(Z)

T, t - push half the trace of period matrix; of instantaneous Z

P, p - pop stack, repeat top element

c - clip the number atop the stack

F - detect the classically forbidden region

Q - quit for large coordinates

v - push the independent variable

n - push the numerical datum

1.3.3 Parameter definition and adjustment

E, e - set the energy, increment the energy

M, m - set the mass, increment the mass (Dirac equations)

W, w - set the weight, increment the weight (Mathieu equations)

Vk - choose the k^{th} potential

1.3.4 Runge-Kutta machinery

u - set solution matrix equal to unit matrix

o - set the Runge-Kutta independent variable origin

D, d - set the Runge-Kutta step size, multiply by a factor

r, R - advance by one Runge-Kutta step (4^{th} or 6^{th} order)

1.3.5 pen movement operators

Ck - set pen color to k

S - set origin, increment for the pen

U - set overall y-offset for the graph

A, a - zero multiple line offset, increment offset

G, H - move to first point, normal or hidden

g, h - move to additional point, normal or visible

s - push, then increment the pen position

There are several documents available explaining REC [1], including a read.me file in the rec subproject of each of the REC files.

1.4 The NeXTSTEP operating system

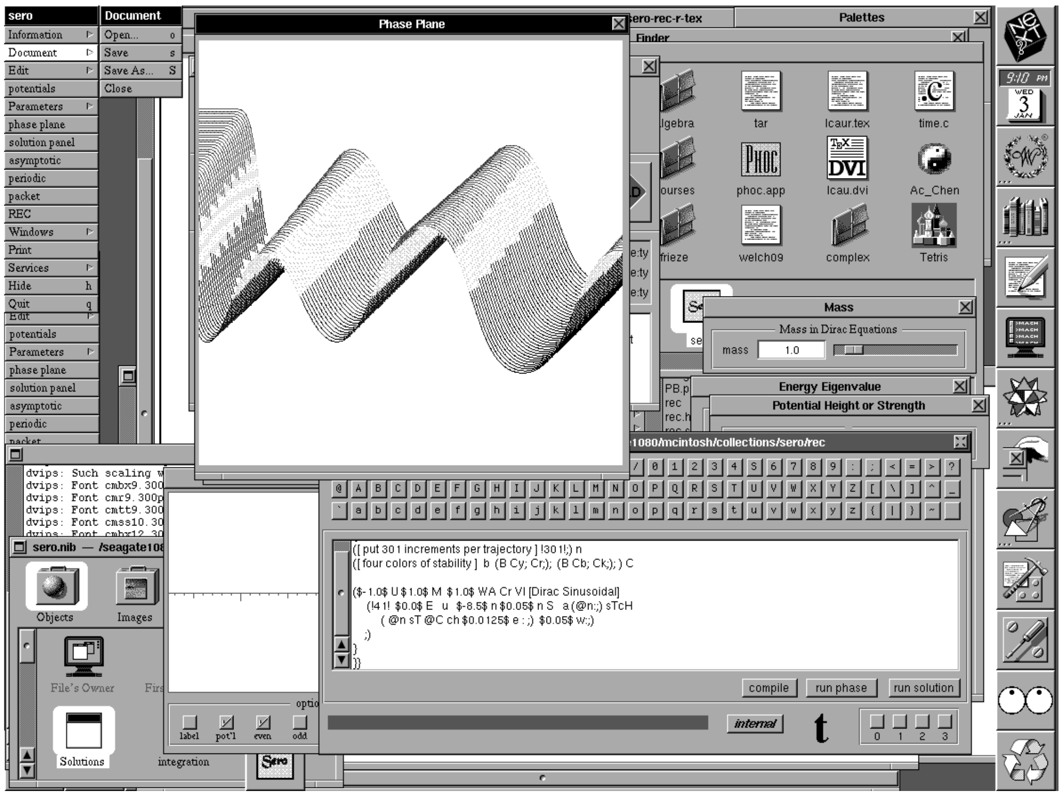
One of the principal advantages of the NeXTSTEP operating system is its use of the PostScript language for all input and output. *All* includes displaying information on the monitor, reading and writing files from disk, the expected task of printing files on a printer, and even includes transmitting them by modem. Amongst the agreeable conveniences is not having to struggle with a screen dump to save information which was originally presented visually.

The appearance which NeXTSTEP presents to the user is via a *File Viewer* which is a window filled with either icons or text, as the user prefers. Moving a cursor by mouse (or sometimes keyboard arrows) indicates a choice of file, each one of which responds in its own manner to clicking or dragging. If it is a subdirectory, additional details are shown; if it is a text file, a page will be opened for editing, while programs will simply be loaded and executed.

Besides the *TextView* window, almost always overlaid as soon as programs begin to execute, there is room around the margins for icons reminding the user of programs and text files which have been held in abeyance, for frequently used programs, and the like. Some space is reserved in the upper left hand corner for the operational menu of the program being executed.

In addition to the mere operating system, which loads and executes programs and interchanges information between different locations, there is an extensive collection of service programs, ranging from language compilers to spelling checkers and even a copy of the Holy Bible. Two of the more important utilities are the *Program Manager* and the *Interface Builder* . There is quite an art to filling up the margins of the screen with useful or potentially useful artifacts while still preserving some space in which to maneuver.

The *Program Manager* mostly checks dates on files, to be sure that the source code for a program is not more recent than the object code. To do so, it maintains lists of all kinds of associated material such as header files, icon images, sounds, and what not. The *Interface*



```
dvips: Such scaling w
dvips: Font cmbx9.300
dvips: Font cmr9.300p
dvips: Font cmtt9.300
dvips: Font cmss10.30
dvips: Font cmbx12.30
```

```
([ put 301 increments per trajectory ] !301!;) n
([ four colors of stability ]  b  (B Cy; Cr;);  (B Cb; Ck;); ) C

($-1.0$ U $1.0$ M  $1.0$ WA Cr VI [Dirac Sinusoidal]
    (!41!  $0.0$ E   u   $-8.5$ n $0.05$ n S   a (@n:;) sTcH
        ( @n sT @C ch $0.0125$ e : ;)  $0.05$ w:;)
   ;)
}
}}
```

Figure 2: A REC-R program, running against a background consisting of the *File Viewer* and some other programs. The dock for permanent programs is visible as a column in the right margin; the menu for the running program occupies a column at the left margin, although it could be moved elsewhere if desired. A corner of *File Viewer* sits at the upper right, the main window for *Interface Builder* sits in the background at the lower left.

Builder permits non-verbal programming, in the sense that windows can be created, stocked with assorted artifacts, and have them interconnected, all through the suitable interpretation of mouse movements. Much programming can be accomplished in those terms, at least during the stages of initial layout.

Going beyond simple tinkering, one needs to use the language Objective C which is a mixture of C^{++} and SmallTalk. The latter uses *Methods*, which are statements with the syntax `[destination operation xxx :argument1 xxx :argument2 ...]`, where `xxx` denotes arbitrary intercalcated text, usually taking the form of helpful comments or comprising pieces of the body of a simple declaration in which the arguments are embedded. The `destination` is an *Object*, which is really a fancy name for a program to be executed[1]. SmallTalk envisioned

[1] *Objects* are actually programs packaged together with data and data structures, independecizing them from one another.

parallel programming, in which several independent programs could communicate with one another by exchanging messages coordinating their activities.

A familiarity with Objective C will be necessary to follow the program listings which are about to be described. In the process it cannot be avoided noticing that NeXTSTEP already includes a vast number of predefined methods destined to manage the user interface. They are all located in a directory, together with supporting information, which can be consulted online, or selectively printed out for further enlightenment.

2 Program Appearance

Given that one is programming in a “graphical user interface” environment (gui), attention has to be given to screen layout and the positioning of labels, controls, and decorations on the screen. These activities can easily absorb much more time and effort than they are worth, but conversely they can fall prey to the routine use of an overly limited selection of icons.

2.1 the .nib window

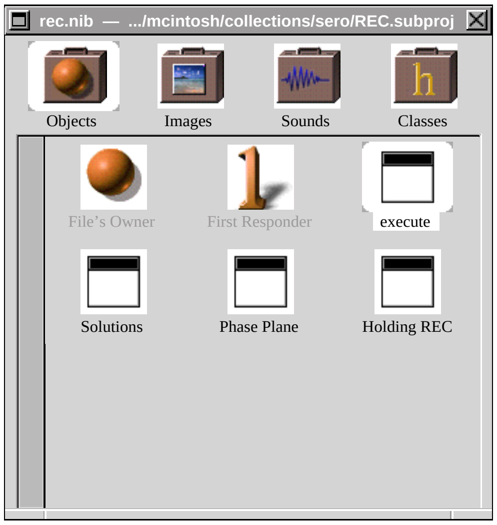


Figure 3: Main window for REC-R.nib, showing the composition of REC-R.

In any event, the stating point is the main window of the *Interface Builder* , which carries representatives of all the windows, panels, and object definitions accessible to the program under construction. In Figure 3 we find an image of this main window, which will always be present during interface construction. Even subprograms get the same attention that a main program would, although naturally the meaning of “File’s Owner” and a few other details would be different.

The .nib window normally sits inconspicuously at the lower left hand corner of the screen, out of the way of any panels or windows being constructed, which are always built full size.

Sizing bars and header bars are available everywhere – on the .nib window as well as the on the items under construction – so that things can be made at one size and position, to be rescaled and relocated before closing the final on their final version.

Beginning with the .nib window as an index, let us examine the contents of the REC-R interface, an inspection which reveals six objects. Two of them, “File’s Owner” and “First Responder,” are standard components of all constructions of the *Interface Builder* , while the other four comprise the material specific to REC-R.

Locating “File’s Owner” is important for implementing delegated methods such as “appDidInit” from the level of the operating system, and for communicating with the main program at the level of subprograms. “First Responder” is less used but is handy for things like communicating with editors.

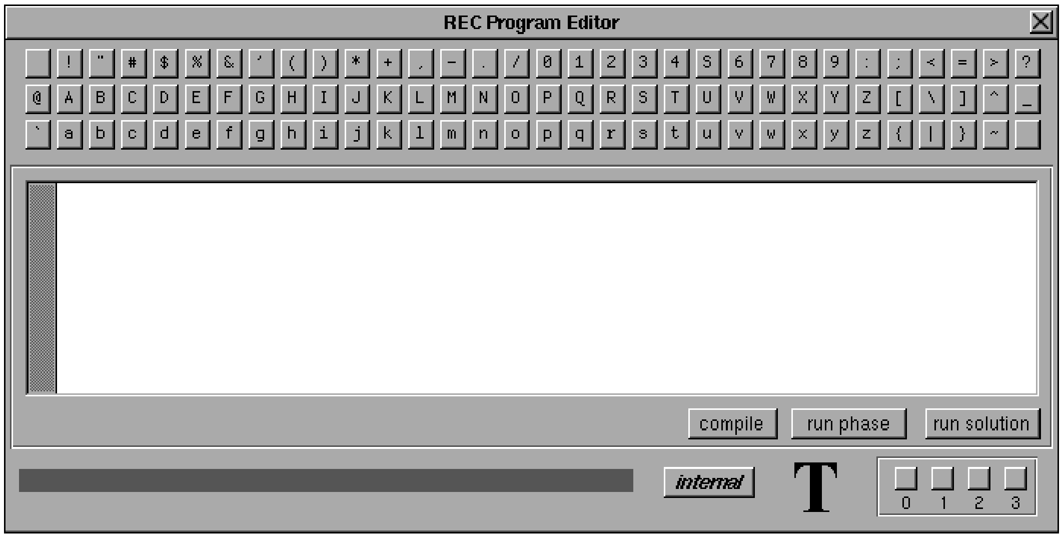

REC Program Editor
! " # $ % & ' ( ) * + , - . / 0 1 2 3 4 5 6 7 8 9 : ; < = > ?
@ A B C D E F G H I J K L M N O P Q R S T U V W X Y Z [\ ] ^ _
` a b c d e f g h i j k l m n o p q r s t u v w x y z { | } ~
compile
run phase
run solution
internal
T
0 1 2 3

Figure 4: The REC-R main window, which contains the REC directory, a program definition area, and a viewer for the resulting construction.

A user of REC-R will deal mostly with three windows, the main window and the two graphics display windows. The main window, labelled “execute” in Figure 3, comes with two alternative forms, which are kept in the panel “Holding REC.” The difference between them is that the browser panel can be exchanged for a text panel, into which programs can be read from disk storage, or composed on the spot. Traffic goes both ways, because the contents of the window can be transferred to the disk, either as a new file or as a correction to the existing file. The text handling variant is shown in Figure 4. The button at the bottom of the main panel, displaying either the title “use external source,” to load the text insert, or “load internal source” to load the browser insert, governs the transition between the two forms.

The window shown in Figure 4 is not the one to be encountered as “execure” while using the *Interface Builder* . The latter differs by having an empty box into which appropriate Views may be copied, which is a way of reducing congestion on a decidedly finite screen. The alternatives are kept in “REC Holding” which will never be shown to the user.

To eliminate the bother of opening disk files, it may be worthwhile to insert some REC programs in the source code for SERO, or preferably in REC-R itself, and recompile it. The result would be a local, nonstandard version of SERO, but could be amply justified for a potential or programming environment which was in extensive use. Another use of the browser option is to see the entire list of REC operators and predicates without having to go to the source code, or view then one by one on through the information buttons on the main panel.

The reason for having two graphics windows is mainly historical, since SERO itself has two different windows. In SERO, each has its own assortment of control buttons. The display of the solution as a function of the variable is long and rectangular because the solution may be of limited height but cover a long interval. The phase plane, on the other hand, wants to be mostly square.because of the similarity of the two components which it displays.

In REC-R, the phase window carries the optional resizing bar, so that it alone would probably suffice.

2.2 Text windows

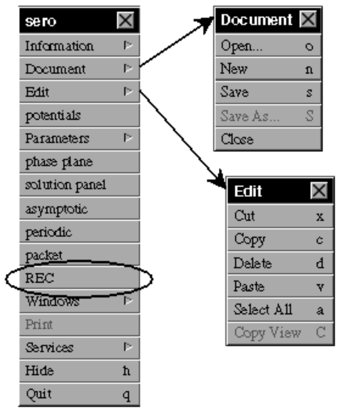


Figure 5: REC-R can use SERO's menu to open .rec-r files which were either created for or destined to be used by, the program window in REC-R. At the same time, the way that REC is called as a subroutine in SERO can be seen in the corresponding menu item. Editing of w text field works without further ado, but the "Copy View" alternative, usually inactive, can copy graphs when the graph windows are active.

There are some standard procedures for incorporating text windows in a programmer's own View, which are probably best obtained by copying them from a program which already has them, and then making appropriate changes in file names, storage arrays, and options. Lacking examples or desiring better information, the book of Garfinkel and Mahoney [3] can be consulted. The tricky aspect of the process is that files have to be opened on at least two levels, first in the MACH or C level, and then again on the NeXTSTEP level; correspondingly they must be closed in stages, running along in reverse order just like balancing parentheses.

Any file manipulation should be connected to the item "Document" in the program's menu, which can be seen in Figure 5, so as to preserve the uniformity in appearance of NeXTSTEP programs.

All text fields are automatically connected to an editor. Besides performing the expected functions of inserting, deleting, and searching, they can share information with other programs through a buffer called the pasteboard. It communicates with editors running in other windows or even from different programs. Besides responding to the keyboard, editing can be accomplished by selecting options in the “Edit” item on the main menu, once having wiped the editing I-beam cursor over a portion of text.

If the “Services” item has been incorporated in a program menu (which has not been done in REC-R), it is possible to have still more elaborate communication between programs, passing selected text either as data or as program instructions.

Of course, there is no reason that a REC program cannot be placed in some file using the text editor the regular way, and subsequently executed. The advantage of editing from the REC program itself is that the code can be tested immediately, and saved only after it is performing satisfactorily.

Supposing that a program is to be loaded from the disk, the “Document” line in the SERO menu will process the file in the standard way, by opening a secondary browser window for disk files. Of course, it will only do that because the necessary programs had already been included in RECView.m and were linked when the interfaces were constructed.

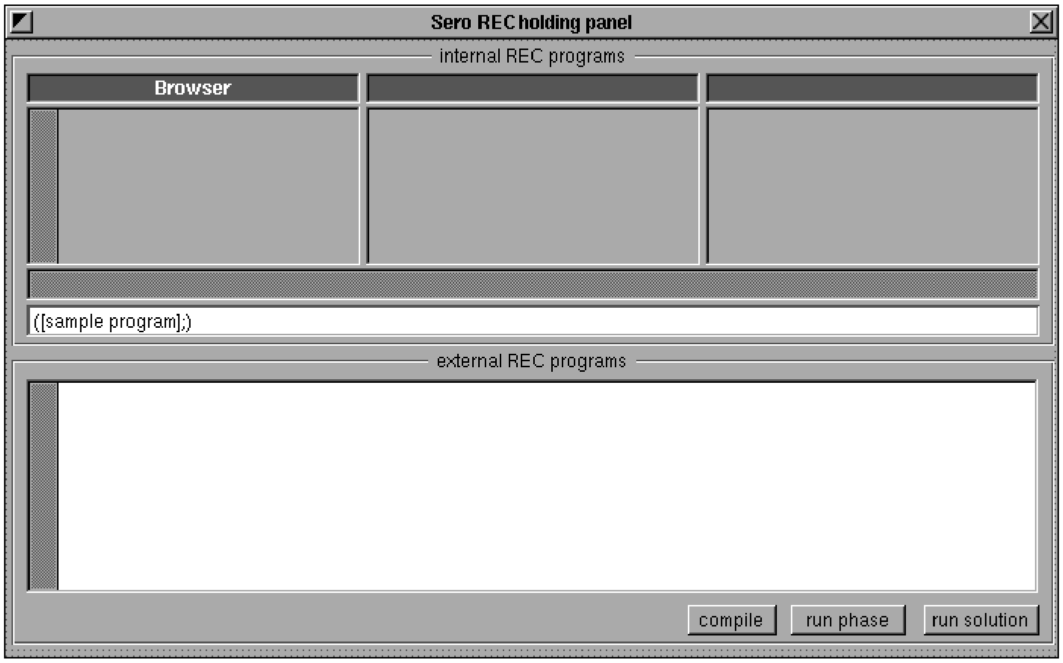
2.3 Browsers

Browsers are another aspect of the operating system which can be incorporated in application programs at the programmer's desire; the procedure is much the same as for coupling text fields to files; namely to copy a browser which is already working, or to look it up in a textbook. The new browser program has to supply the text (or even icons) that the browser will display, which is accomplished by delegation. A delegate can be associated with an object by mouse dragging in the interface builder, or set up through programming.

A *delegate* is expected to provide methods which could have been incorporated in the delegating program, but for some reason, weren't. The principal reason is that there was no way of forseeing, when the master program was written, exactly what the methods were supposed to do. After all, each application will have its own items to display, and its own reactions to their selection; such information is best incorporated in the application program itself.

Looking around the REC-R main window, another text field can be seen; one which is used for holding error messages and comments, and especially, displaying the definition of REC operators. In fact, there is a whole button array bearing the three printable quadrants of the ASCII alphabet, whose sole purpose is to provide these reminders to the user. Although there is no requirement within the language to do so, it is customary to use single letters (or ASCII characters) for the operators because of their convenience and ease of remembering. Of course, such a principle only works when there are enough letters to go around, which is one of the reason for tailoring individual REC programs instead of concocting a universal or general purpose REC.

The browser in the main window of REC-R is stocked with information of various types. The first item copies the header file which identifies the operators and predicates in REC-R, a variant of which has to be included in all REC programs. It supplements the button panel for REC definitions, because it is easier to scan a list of possibilities than to press the buttons



([sample program];)

Figure 6: The REC holding window, which contains two alternative Views, one at a time of which will be placed in the main window..

one by one. On the other hand, for quick recollection or to identify an unfamiliar letter in a displayed program, the buttons are handier.

There are benefits to be had in stocking the browser with some simple and typical programs, so that REC-R can be tried out at once without having to worry about disk files or other extraneous factors.

2.4 Phase window, solution window

Drawing pictures brings up a series of problems, not all of which are evident at first. Scale and centering are among the most obvious, easily compensated by including provisions for moving the figure and changing its size. Conflicts can still remain, because excessive compression may make the figure difficult to read, or because the final result has to be presented on a sheet of paper of given size. The next remedy, but which REC-R does not use, is to divide a figure into panels, or use scrolling, at the price of increased programming complexity.

Rotation and reflection bring up new challenges, although some foresight can accomodate them as well. The simplest solution to figure revision is to introduce homogeneous coordinates, and subject all drawing operations to a projective transformation. Of course that implies matrix multiplication, which may or may not have a deleterious effect on performance. Computers are now sufficiently rapid that simple drawings can receive a fair amount of preprocessing while producing results compatible with human reaction time, which means about twenty frames per second.

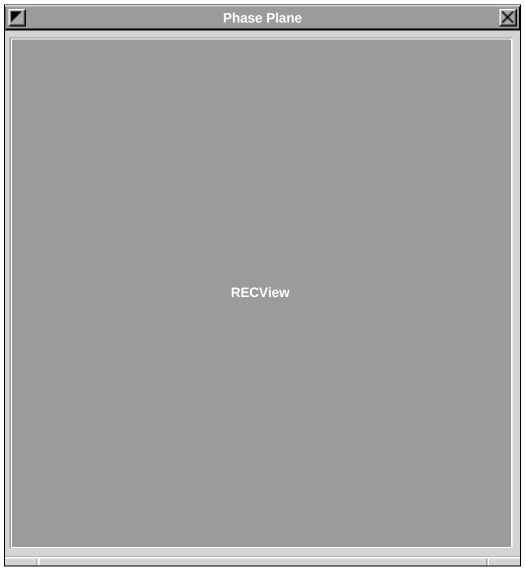


Figure 7: The phase window can be enlarged or reduced by using the window sizer on its bottom margin.

While projective facilities are easy enough to incorporate in newly written programs, they are noticeably lacking in such existing service programs as Draw with which the new programs may still want to interact.

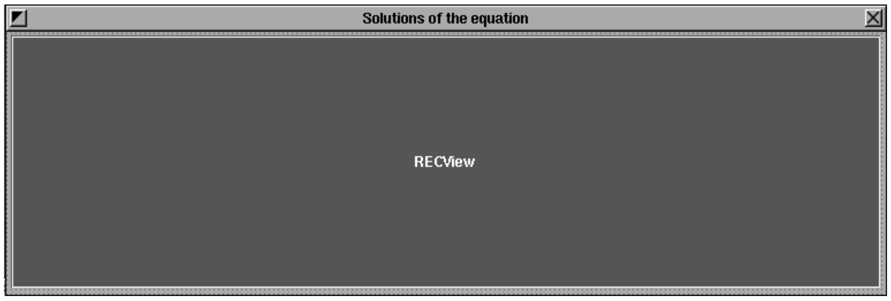


Figure 8: The solution window is inflexible.

For the moment, REC-R provides two parallel views, one in the phase window, shown in Figure 7, the other in the solution window shown in Figure 8. Both contain an image which may be copied into the pasteboard by invoking the “Copy View” button on the main menu (as seen in Figure 5) and then transferred into Draw by pasting. But so far, they lack any further processing.

3 The REC Header and Program Listing

Besides other headers, such as `rec.h` which serves the REC compiler, and headers germane to a particular application, one more should be provided which is devoted exclusively to defining the REC operators and predicates which are going to be used. Additionally, the operators and predicates themselves must be defined, more likely in C than not; keeping them all together in one place facilitates program maintainance.

3.1 REC header

The header file seems excessively long when only a handful of definitions are actually stated, but in keeping with the tradition of using single ASCII characters as symbols, conserving the full list helps browsers and inspectors work uniformly.

Moreover, insertions or deletions to the table are confined to the locations already alloted for them. Such systematics often reduce error and increase programming convenience. In fact, for some of the more common variants, such as floating point numbers, comments in the form of quoted strings, or frequently used operators, including both forms in the table with the unused line neutralized by making it into a comment can reduce confusion and save programming effort.

This header file is also the place to declare all the REC functions which it introduces — at least the compiling and executing functions — if not all their satellites. Those can probably be accounted for in the REC implementation file with less overall effort.

```
/* rectbl.h -- G. Cisneros, 4.91; rev. 2.10.91 */
/* Mandatory declarations */

# include "rec.h"

int r_lpar();       /* compilation of a left parenthesis            */
int r_rpar();       /* compilation of a right parenthesis           */
int r_colon();      /* compilation of a colon                       */
int r_semicol();    /* compilation of a semicolon                   */
int r_code();       /* compilation of a simple operator             */
int r_oper1();      /* compilation of an operator with an ASCII arg */
int r_pred();       /* compilation of a simple predicate            */
int r_pred1();      /* compilation of a predicate with an ASCII arg */
int r_noopc();      /* compilation of no-op                         */
int r_comment();    /* compilation of a (bracket enclosed) comment  */
int r_ubrack();     /* compilation of an unbalanced right bracket   */
int r_lbrace();     /* compilation of a (brace enclosed) program    */
int r_ubrace();     /* compilation of an unbalanced right brace     */

int r_call();       /* execution of predicate @x (call subr. x)     */
int r_quit();       /* execution of operator _(exit to O.S.)        */
int r_xbrace();     /* execution of brace-enclosed program          */

/* Optional: Counter predicate */

 int r_ctrc();            /* compilation of a counter */
 int r_ctrx();            /* execution of a counter   */

/* Optional declarations, needed only in versions using quotes and
   numbers (see cnum.c and cquo.c); r_ld, r_ldch and r_xstr must
   be provided by user */

// int r_dquote(); /* compile doubly quoted string into symbol table      */
// int r_squote(); /* compile singly quoted char into the program array   */
// int r_cmin();   /* compile '-' as simple operator, call r_cnum if followed
//              by digit or period */
// int r_cnum();   /* compile a number (WORD, LONG or REAL) into sym. table */

// int r_ld();            /* execution of compiled numbers */
// int r_ldch();      /* execution of apostrophe (single quote op.) */
// int r_xstr();      /* execution of quoted strings */

int r_cdblp(), r_ldblp();

/* Declarations of user-provided execution subroutines */

void ropla(), roplc(), ropld(), rople(), roplf(), roplg(), roplh();
void ropli(), roplj(), roplk(), ropll();
void roplm(), ropln(), roplo();
void roplp(), roplq(), roplr(), ropls(), roplu(), roplv();

void ropua(), ropuc(), ropud(), ropue();
void ropuf(), ropug(), ropuh();
void ropui(), ropuj(), ropuk(), ropul();
void ropum(), ropun(), ropuo();
void ropup(), ropur(), ropus(), roput();
void ropuv();

int  rpruf(), rpruq();

void ropca(), ropqm();

int  rprns(), rprps();
int  rprze(), rpron(), rprtw(), rprth();

/* Table of compiling/executing function definitions,
   a pair for each printing ASCII character */

struct fptbl dtbl[] = {
  r_noopc,   FALSE,  " space (separator)",                                /* [blank] */
  r_ctrc,  r_ctrx,   " ! - count to n, as in (!n![repeat]:[finish];)",           /* ! */
  r_noopc,   FALSE,  "",                                          /* " quoted string */
  r_pred,    rprns,  " # - rand pred, prob 1/2, as in (#[heads];[tails];)  ",     /* # */
  r_cdblp,   r_ldblp," $ - floating point number, sits until changed, e.g. $2.5$",/* $ */
  r_pred,    rprps,   " % - rand pred, prob 1/10, as in (%[common];[rare];)",     /* % */
  r_noopc,   FALSE,   " & -",                                                     /* & */
  r_noopc,   FALSE,   "",                                         /* ' quoted char */
  r_lpar,    FALSE,   " ( - start of expression",                               /* (  */
  r_rpar,    FALSE,   " ) - end of expression",                                 /* )  */
  r_noopc,   FALSE,   " * -",                                                     /* * */
  r_noopc,   FALSE,   " + - ",                                                    /* + */
  r_noopc,   FALSE,   " , - ",                                       /* , separator */
  r_noopc,   FALSE,   " - - ",                                                    /* - */
  r_noopc,   FALSE,   " . - ",                                    /* . f.p. number */
  r_noopc,   FALSE,   " / - ",                                                    /* / */
  r_pred,    rprze,    " 0 - ",                                         /* 0 number */
  r_pred,    rpron,    " 1 - ",                                         /* 1 number */
  r_pred,    rprtw,    " 2 - ",                                         /* 2 number */
r_pred,    rprth,    " 3 - ",                                             /* 3 number */
r_noopc,   FALSE,    " 4 - ",                                             /* 4 number */
r_noopc,   FALSE,    " 5 - ",                                             /* 5 number */
r_noopc,   FALSE,   " 6 - ",                                              /* 6 number */
r_noopc,   FALSE,   " 7 - ",                                              /* 7 number */
r_noopc,   FALSE,   " 8 - ",                                              /* 8 number */
r_noopc,   FALSE,   " 9 - ",                                              /* 9 number */
r_colon,   FALSE,   " : - repeat from opening lparen ",               /* : iteration */
r_semicol, FALSE,   " ; - exit at terminating rparen with value T  ", /* ; true exit */
r_noopc,   FALSE,   " < - ",                                                     /* < */
r_noopc,   FALSE,   " = - ",                                                     /* = */
r_noopc,   FALSE,   " > - ",                                                     /* > */
r_code,    ropqm,   " ? - print register values ",                               /* ? */
r_pred1,   r_call,  " @ - call a subroutine, e.g. @x calls x ", /* @ subroutine call */
r_code,    ropua,   " A - make the point offset equal to zero ",                 /* A */
r_pred,    rprub,   " B - does the top of the stack lie between -1.0 and 1.0?", /* B */
r_oper1,   ropuc,   " C - set color with Cx, x = r,g,b,c,m,y,k,w ",             /* C */
r_code,    ropud,   " D - set the Runge-Kutta step size, as in $0.1$ D",        /* D */
r_code,    ropue,   " E - set the Energy, as in $1.5$ E, or in s E ",           /* E */
r_pred,    rpruf,   " F - relative signs in coeff. (F[forbidden];[not];) ",     /* F */
r_code,    ropug,   " G - move to first graph point ",                           /* G */
r_code,    ropuh,   " H - record first point in a hidden line sequence ",       /* H */
r_code,    ropui,   " I - push [0][0] element of coefficient matrix ",          /* I */
r_code,    ropuj,   " J - push [0][1] element of coefficient matrix ",          /* J */
r_code,    ropuk,   " K - push [1][0] element of coefficient matrix ",          /* K */
r_code,    ropul,   " L - push [1][1] element of coefficient matrix ",          /* L */
r_code,    ropum,   " M - set mass for Dirac Equations, e.g. $1.0$M ",          /* M */
r_code,    ropun,   " N - ",                                                     /* N */
r_code,    ropuo,   " O - set origin, incr of ind var: $-7.5$ n $0.1$ n O, say",/* O */
r_code,    ropup,   " P - pop the stack ",                                       /* P */
r_pred,    rpruq,   " Q - quit for large coordinates",                           /* Q */
r_code,    ropur,   " R - advance by one 6th order Runge-Kutta step",           /* R */
r_code,    ropus,   " S - initialize step variable, as in $-7.5$ n $0.1$ n S ", /* S */
r_code,    roput,   " T - push half the trace of the period matrix",            /* T */
r_code,    ropuu,   " U - Offsets the y-origin, as in $-1.5$ U",                /* U */
r_oper1,   ropuv,   " V - Vn choose potential n [e.g. V5=harmonic oscillator]", /* V */
r_code,    ropuw,   " W - define the weight coefficient, as in $1.0$ W ",       /* W */
r_noopc,   FALSE,   " X - ",                                                     /* X */
r_noopc,   FALSE,   " Y - ",                                                     /* Y */
r_noopc,   FALSE,   " Z - ",                                                     /* Z */
r_comment, FALSE,   " ] - begin comment, as in [comment] ",                     /* [ */
r_noopc,   FALSE,   " \\ - ",                                                    /* \ */
r_ubrack,  FALSE,   " ] - end comment   ",                                       /* ] */
r_code,    ropca,   " ^ - ",                                                     /* ^ */
r_code,    r_quit,   " _- ",                                           /* _ panic exit */
r_noopc,   FALSE,   " ‘ - ",                                                     /* ‘ */
r_code,    ropla,   " a - add an ever increasing offset to point coordinates",  /* a */
r_pred,    rprlb,   " b - is the top of the stack positive?",                   /* b */
r_code,    roplc,   " c - clip the number on top of the stack",                 /* c */
r_code,    ropld,   " d - rescale the step size, as in $1.125$ d ",             /* d */
r_code,    ople,    " e - modify the energy, as in $-0.01$ e ",                 /* e */
r_code,    roplf,   " f - ",                                                     /* f */
r_code,    roplg,   " g - graph additional points",                              /* g */
r_code,    roplh,   " h - Graph what is visible over (& under) the horizons ",  /* h */
r_code,    ropli,   " i - push [0][0] element of solution matrix",              /* i */
r_code,    roplj,   " j - push [0][1] element of solution matrix",              /* j */
r_code,    roplk,   " k - push [1][0] element of solution matrix",              /* k */
r_code,    ropll,   " l - push [1][1] element of solution matrix",              /* l */
```

```
  r_code,    roplm,   " m - modify the mass, as in $-0.01$ m ",                 /* m */
  r_code,    ropln,   " n - push the numerical datum [$1.0$ n, for example]",   /* n */
  r_code,    roplo,   " o - set Runge-Kutta variable's origin: $0.0$ o, say ",  /* o */
  r_code,    roplp,   " p - push the top of the stack ",                        /* p */
  r_code,    roplq,   " q - ",                                                 /* q */
  r_code,    roplr,   " r - advance by one 4th order Runge-Kutta step",        /* r */
  r_code,    ropls,   " s - push, then increment, the step variable ",         /* s */
  r_code,    roplt,   " t - push half the trace of the solution matrix",       /* t */
  r_code,    roplu,   " u - set solution matrix equal to unit matrix",         /* u */
  r_code,    roplv,   " v - push the independent variable",                    /* v */
  r_code,    roplw,   " w - increment the weight, as in $0.1$ w",              /* w */
  r_noopc,   FALSE,   " x - ",                                                 /* x */
  r_noopc,   FALSE,   " y - ",                                                 /* y */
  r_noopc,   FALSE,   " z - ",                                                 /* z */
  r_lbrace,  r_xbrace," { - start program, e.g. {(...) a (...) b ... (...)}", /* { */
  r_noopc,   FALSE,   " | - ",                                                 /* | */
  r_ubrace,  FALSE,   " } - end of program ",                                  /* } */
  r_noopc,   FALSE,   " ~ - "                                                  /* ~ */
};
```

3.2 REC-R data flow

Figure 1 shows the data flow in REC/R, which is not very extensive. There is essentially one array, of ample dimension, which will hold the coordinates of the line segments which are going to be drawn. Rather than define a segment by its endpoints, only the terminal point for each segment is provided. Since an initial point has to start the sequence, it is provided by a different operator. For ease of remembrance, it is the capital letter corresponding to the ongoing movement; remember that sentences start with capital letters.

Although simple graphs suffice to show solutions, either as functions or in the phase plane, they are hardly complicated enough to justify the construction of a whole elaborate program such as REC-R. Rather, it is the graphing of multiple solutions, usually obtained as the result of varying some parameter or other, which justifies all the additional effort.

Superposing multiple curves is less and less informative as the number of curves increases, but offsetting them from one another helps to maintain their identity. Even so, as the curves cross back and forth over one another, the overall graph can become quite confusing, which is the point at which the corves might be considered as having been inscribed on an opaque surface of which only the visible portion is drawn.

This was one of the first tasks given to computer graphics, whose solution was to maintain horizons as auxiliary curves and then to display only maxima and minima in comparison with the horizon, rather than the data itself. To do this well implies calculating good intersections of the data with the horizons, which implies adding these intersections to the list of function arguments.

REC-R does not take this step, but has provision for checking values relative to the horizon and storing the new horizon. The operators `G` and `g` produce ordinary pen movements, `H` and `h` take into account the horizone. The whole horizon apparatus is initialized by `A`, while `a` generates successive offsets.

Since graphing is independent of Runge-Kutta integration, there are two variables to be kept track of. The first, the horizontal coordinate of the graph and its increment, are initialized by `S`. The increment is applied, one step at a time, by `s`. Vertical positioning is adjusted by `u`, to get a constant y-offset.

The second variable is the one belonging to the independent variable of the Runge-Kutta integration, which is initialized by `o` and incremented automatically by one of the integration operators `r` or /tt R. Its value, when needed, can be placed on the pushdown stack by `v`. That might be done when the graphical and integration step were equal, since the program which does the actual plotting, the PostScript functions `moveto` and `lineto` gets arguments which take into acount the physical dimensions and resolution of the screen, as well as typical ranges of potentials.

However, very little has been done to incorporate variable scaling or changes of coordinates into the pen movement operators. One facility which has been provided, however, is the use of the hyperbolic tangent to compress the range of a variable with the clipping operator `c`.

Aside from operators related to presenting the graph, there are others related to producing the data, which are all related to the step by step development of the numerical integration. Accordingly individual elements of either the coefficient matrix or the solution matrix, can be brought to the pushdown list for graphing. Beyond that, the potentials included in SERO as examples all have parameters, whose values can be set and incremented according to the desired graph.

3.3 REC operators and predicates

The ultimate authority which determines the action of the REC-R operators and predicates is the `C` program `rec-sero.c`, whose verbatim listing follows:

```
#import "rec.h"
#import "sero.h"
#import "rectbl.h"
#import <math.h>
#import  <dpsclient/psops.h>
#import  <appkit/graphics.h>

# define HZAR    500 /* array length for graph horizons */

int    options[4]={FALSE,FALSE,FALSE,FALSE};

extern double r_dblpar; /* value of REC's $xxx.yy$ */

int    stackptr=-1;
int    (*RECPot)()=&potl5;
int    RECPotl[20] = {
      &potl1,  /* free particle  */
      &potl1,  /* free particle  */
      &potl2, /* finite well */
      &potl3, /* barrier */
      &potl4, /* step */
      &potl5, /* harmonic oscillator */
      &potl6, /* quartic oscillator */
      &potl7, /* factored dirac h.o. */
      &potl8, /* gaussian dirac eqn */
      &potl9, /* Mathieu equation */
      &potl10, /* linear potential */
      &potl11, /* coulomb potential */
      &potl12, /* inverse square */
      &potl13, /* damped harmonic osc */
      &potl14, /* dirac harmonic osc */
      &potl15, /* factored inv. square */
      &potl15, /* factored inv. square */
      &potl15, /* factored inv. square */
      &potl18, /* dirac harmonic osc */
      &potl19 /* dirac periodic potl */
      };

int    RECOffset = 0;
double yoffset   = 0.0;
double RECEnergy = 0.5;
double RECMass   = 1.0;
double RECWeight = 1.0;
double RECGPoint = 0.0;
double RECGStep  = 0.1;
double RECKPoint = 0.0;
double RECKStep  = 0.1;
double RECPtntl[4];
double RECSolun[4];
double RECPush[10];
double horhi[HZAR];
double horlo[HZAR];
double lastex, lastwy;

int    horix=0;
extern int ptj;
extern double hi, em, egy;

/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */
/*            definition of REC operators and predicates                  */
/* - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - */

int rprze() {return(options[0]==TRUE);}
int rpron() {return(options[1]==TRUE);}
int rprtw() {return(options[2]==TRUE);}
int rprth() {return(options[3]==TRUE);}

void ropla() {RECOffset+=1; horix=0;}  /* increase offset, reset horizon */

int rprlb() {return RECPush[stackptr]>=0.0;} /* top of stack is positive */

void roplc() {RECPush[stackptr]=2.0*tanh(0.5*RECPush[stackptr]);}

void ropld() {RECKStep*=r_dblpar;} /* modify R-K step size */

void rople() {RECEnergy+=r_dblpar;} /* modify Energy */

void roplf() {} /* shorten step    */

void roplg() {double ex, wy; /* graph additional points */
if (stackptr >= 10 || stackptr < 1) return;
wy=RECPush[stackptr--]+((double)RECOffset)*RECGStep+yoffset;
ex=RECPush[stackptr--]+((double)RECOffset)*RECGStep;
videoplot(ex,wy,1);
}

void roplh() {double x, y, z, ex, wy; int i; /* graph additional points */
if (horix==0) return;
if (stackptr >= 10 || stackptr < 1) return;
wy=RECPush[stackptr--]+((double)RECOffset)*RECGStep+yoffset;
```

```
ex=RECPush[stackptr--]+((double)RECOffset)*RECGStep;
i=horix+RECOffset;
if (wy>=horhi[i]) {
    horhi[i]=wy;
    if (lastwy>=horhi[i-1]) {
videoplot(lastex,lastwy,0);
videoplot(ex,wy,1);
    } else {
z=(horhi[i]-horhi[i-1]-wy+lastwy);
if (z==0.0) x=lastex; else
   x=lastex+(RECGStep*(lastwy-horhi[i-1]))/z;
y=lastwy+(x-lastex)*((wy-lastwy)/RECGStep);
videoplot(x,y,0);
videoplot(ex,wy,1);
        }
    }
if (wy<horhi[i] && lastwy>=horhi[i-1]) {
    z=(horhi[i]-horhi[i-1]-wy+lastwy);
    if (z==0.0) x=lastex; else
        x=lastex+(RECGStep*(lastwy-horhi[i-1]))/z;
    y=lastwy+(x-lastex)*((wy-lastwy)/RECGStep);
    videoplot(lastex,lastwy,0);
    videoplot(x,y,1);
    }
if (wy<horlo[i]) {
    horlo[i]=wy;
    if (lastwy<horlo[i-1]) {
videoplot(lastex,lastwy,0);
videoplot(ex,wy,1);
    } else {
z=(horlo[i]-horlo[i-1]-wy+lastwy);
if (z==0.0) x=lastex; else
   x=lastex+(RECGStep*(lastwy-horlo[i-1]))/z;
y=lastwy+(x-lastex)*((wy-lastwy)/RECGStep);
videoplot(x,y,0);
videoplot(ex,wy,1);
    }
}
if (wy>horlo[i] && lastwy<=horlo[i-1]) {
    z=(horlo[i]-horlo[i-1]-wy+lastwy);
    if (z==0.0) x=lastex; else
        x=lastex+(RECGStep*(lastwy-horlo[i-1]))/z;
    y=lastwy+(x-lastex)*((wy-lastwy)/RECGStep);
    videoplot(lastex,lastwy,0);
    videoplot(x,y,1);
    }
lastex=ex; lastwy=wy;
if (horix<HZAR-1) horix++;
}

void ropli() {
stackptr++;
if (stackptr>=10) return;
RECPush[stackptr]=RECSolun[0];
} /* j[0][0] solution element       */

void roplj() {
stackptr++;
```

```
if (stackptr>=10) return;
RECPush[stackptr]=RECSolun[1];
} /* [0][1] solution element       */

void roplk() {
stackptr++;
if (stackptr>=10) return;
RECPush[stackptr]=RECSolun[2];
} /* [1][0] solution element       */

void ropll() {
stackptr++;
if (stackptr>=10) return;
RECPush[stackptr]=RECSolun[3];
} /* [1][1] solution element       */

void roplm() {RECMass+=r_dblpar; em=RECMass;}

void ropln() {
stackptr++;
if (stackptr>=10) return;
RECPush[stackptr]=r_dblpar;
}  /* push the $$ register */

void roplo() {RECKPoint=r_dblpar;}    /* set origin */

void roplp() {
stackptr++;
if (stackptr>=10) return;
RECPush[stackptr]=RECPush[stackptr-1];
}     /* copy top element on stack */

void roplq() {}

void roplr() {serss(
RECSolun,
RECPtntl,
&RECKPoint,
RECKStep,
RECEnergy,
RECPot,
serr4);}
      /* one step of fourth order Runge-Kutta integration */

void ropls() {  /* push, then increment the step variable */
stackptr++;
if (stackptr>=10) return;
RECPush[stackptr]=RECGPoint;
RECGPoint+=RECGStep;
}

void roplt() {
stackptr++;
if (stackptr>=10) return;
RECPush[stackptr]=0.5*(RECSolun[0]+RECSolun[3]);
}        /* half trace of solution matrix = cos phi */

void roplu() {serum(RECSolun);}     /* put unit matrix as solution matrix */
void roplv() {      /* push independent variable */
stackptr++;
if (stackptr>=10) return;
RECPush[stackptr]=RECKPoint;
}

void roplw() {RECWeight+=r_dblpar; hi=RECWeight;}

/* ---- upper case ----- */

void ropua() {int i; /* reset offset for plotted point */
 horix=0;
 RECOffset=0;
 for (i=0; i<HZAR; i++) horhi[i]=-10.0;
 for (i=0; i<HZAR; i++) horlo[i]= 10.0;
}

int rprub() {return (RECPush[stackptr] <= 1.0 && RECPush[stackptr] >= -1.0);}
        /* top of stack lies between -1.0 and 1.0  */

void ropuc() {int c;       /* set color */
PSstroke();
c = *r_pc++;
switch (c) {
  case 'r': PSsetrgbcolor(1.0,0.0,0.0); break;
  case 'g': PSsetrgbcolor(0.0,1.0,0.0); break;
  case 'b': PSsetrgbcolor(0.0,0.0,1.0); break;
  case 'o': PSsetrgbcolor(0.8,0.4,0.1); break;
  case 'v': PSsetrgbcolor(1.0,0.0,1.0); break;
  case 'c': PSsetcmykcolor(1.0,0.0,0.0,0.0); break;
  case 'm': PSsetcmykcolor(0.0,1.0,0.0,0.0); break;
  case 'y': PSsetcmykcolor(0.0,0.0,1.0,0.0); break;
  case 'k': PSsetgray(NX_BLACK); break;
  case 'd': PSsetgray(NX_DKGRAY); break;
  case 'l': PSsetgray(NX_LTGRAY); break;
  case 'w': PSsetgray(NX_WHITE); break;
  default:  PSsetgray(NX_BLACK); break;}
}

void ropud() {RECKStep=r_dblpar;} /* set R-K step size */

void ropue() {RECEnergy=r_dblpar;}       /* set Energy */

int  rpruf() {return (RECPtntl[1]*RECPtntl[2]>=0.0);}      /* coeff parity */

void ropug() {double ex, wy;   /* graph additional points */
if (stackptr >= 10 || stackptr < 1) return;
wy=RECPush[stackptr--]+((double)RECOffset)*RECGStep+yoffset;
ex=RECPush[stackptr--]+((double)RECOffset)*RECGStep;
videoplot(ex,wy,0);
}

void ropuh() {double ex, wy; int i; /* graph first point     */
wy=RECPush[stackptr--]+((double)RECOffset)*RECGStep+yoffset;
ex=RECPush[stackptr--]+((double)RECOffset)*RECGStep;
lastex=ex; lastwy=wy;
i=horix+RECOffset;
```

```
if (wy>horhi[i]) horhi[i]=wy;
if (wy<horlo[i]) horlo[i]=wy;
if (horix<HZAR-1) horix++;
}

void ropui() {
stackptr++;
if (stackptr>=10) return;
RECPush[stackptr]=RECPtntl[0];
} /* j[0][0] solution element       */

void ropuj() {
stackptr++;
if (stackptr>=10) return;
RECPush[stackptr]=RECPtntl[1];
} /* [0][1] solution element       */

void ropuk() {
stackptr++;
if (stackptr>=10) return;
RECPush[stackptr]=RECPtntl[2];
} /* [1][0] solution element       */

void ropul() {
stackptr++;
if (stackptr>=10) return;
RECPush[stackptr]=RECPtntl[3];
} /* [1][1] solution element       */

void ropum() {RECMass=em=r_dblpar;}

void ropun() {}

void ropuo() { /* origin, increment - ind. var. */
if (stackptr >= 10 || stackptr < 1) return;
RECGStep=RECPush[stackptr--];
RECKPoint=RECPush[stackptr--];
}

void ropup() {if (stackptr<-1) {stackptr=-1; return;} else stackptr--;}

int  rpruq() {return(RECPush[stackptr]*RECPush[stackptr]>25.0);}
/* test for minimum scale */

void ropur() {serr6(RECSolun,&RECKPoint,RECGStep,RECEnergy,RECPot);}
   /* one step sixth order Runge-Kutta integration */

void ropus(){
if (stackptr >= 10 || stackptr < 1) return;
RECGStep=RECPush[stackptr--];
RECGPoint=RECPush[stackptr--];
}  /* Initiallize the graphical step */

void roput() {
stackptr++;
if (stackptr>=10) return;
hi=RECWeight;
RECPush[stackptr]=serar(RECEnergy,RECPot,&serr6);
```

```
}     /* half trace of solution matrix over a period */

void ropuu() {yoffset=r_dblpar;} /* offset y-origin */

void ropuv() {int i;
i=*r_pc++;
if (i<='9') i-='0'; else {if (i<='Z') i-='A'-10; else i-='a'-10;}
RECPot=RECPotl[i];
printf("RECPot = %2d\n",i);
} /* choose potential */

void ropuw() {RECWeight=hi=r_dblpar;}    /* define weight */

/* ------------ */

void roppl() {RECKPoint+=RECKStep;} /* advance by one Runge-Kutta sized step */

void ropca() {}

void ropqm() { /* print reg values */
printf("z = %6.3f, E = %6.3f\n",RECKPoint,RECEnergy);
printf("Solution = (%6.3f,%6.3f,%6.3f,%6.3f)\n",
RECSolun[0],
RECSolun[1],
RECSolun[2],
RECSolun[3]);
printf("Potential = (%6.3f,%6.3f,%6.3f,%6.3f)\n",
RECPtntl[0],
RECPtntl[1],
RECPtntl[2],
RECPtntl[3]);
printf("\n");
}

int rprns() {return (int)(random()&01);} /* random 1/2 pred */

int rprps() {return random()%10==1;} /* random 1/10 pred */
```

4 RECView.h Header Listing

Although the headers for a methods file can contain all the things that ordinary headers do, they are mostly given over to listing objects and method prototypes for their associated file. If the Interface Builder has been used without the benefit of a header file, there is an option called “unparsing” which will create one. Conversely, if the text of the header file has been changed, the `.nib` file can be updated by using “unparse.”

It is worth noting that there is no convenient way to supply several arguments to a method which is going to be connected graphically in the Interface Builder. But methods are not supposed to look at each other's data anyway; they should only transmit the data or pointers to the data in response to a polite request. To know who made a request, and thus to whom to send the reply, a method can volunteer its own name which the called program can recognize as an argument called `:sender`, at which time it knows how to ask for further services or data. That is why `:sender` is the only argument of most of the methods in the prototype list, and certainly for those which are going to be connected using *Interface Builder* .

```
#import <appkit/appkit.h>

@interface RECView:View {

id bigView;
id bigWindow;
id RECOButn;
id  RECLine;
id RECMatrix;
id RECPanel;

id sourceView;
id sourceWindow;

id phaseView;
id phaseWindow;
id solutionView;
id solutionWindow;
id controlWindow;

id browser;
id comment;

id bigBox;
id extBox;
id intBox;
id lilBox;

id execWin;
id RECValue;

id menuCopy;
id menuPrint;

char *filename;
char **filenames;

}

- ascii:sender;
- recMatrix:sender;

- loadRECPanel:sender;
- loadRECSource:sender;

- nhul:sender;
- recompile:sender;
- run:sender;
- runPhase:sender;
- runSolution:sender;
- runView:sender;
- runRECLine:sender;
- runRECOffLine:sender;
- useRECLine:sender;

- setOption:sender;

- smartPrintBigWin:sender;
- openBigWin:sender;

- close:sender;
- open:sender;
- setFilename:(const char *)aFilename;
- save:sender;
- saveAs:sender;

- copyREC:sender;
- copyAndRunREC:sender;
- copySomeView:theView;

@end

/* end RECView.h */
```

5 RECView.m Program Listing

The part of REC-R which is strictly REC is contained in the `C` program `rec-sero.c`, but it is embedded in the NeXTSTEP method `RECView.m` and its header, `RECView.h` so as to get the advantage of the whole user interface and operating system. Consequently `RECView.m` consists mainly of boiler plate operating the text files, the browser, copying Views hither and yon and printing them, and so on. Nevertheless it is included to give a complete definition of REC-R. Of course, certain vestiges of its subservience to SERO still remain, but are not elaborated.

```
#import "rec.h"
#import "RECView.h"
#import <string.h>

# define CLEN 1000 /* buffer length for rec source      */
# define PLEN 3000 /* buffer length for rec object      */
# define XLEN    128 /* buffer length for rec predefined   */
# define NY     25 /* number of demonstrations       */

# define PI 3.14159

int    showRECBack=YES;
int    stillUnused=YES;

int    isErrm=(int)FALSE;
char   errm[50];

char   cstr[CLEN]; /* console rec program     */
Inst   rprg[PLEN]; /* rec program array       */
Inst   xprg [10][XLEN];
char   idef [1] [30] = {"(z((!100!0r:;)1r2r(0r:;);) a"};
extern struct fptbl dtbl[];

int    phil; /* length of cstr    */

@implementation RECView

/* = = = = = = = = = = =   R E C / S E R O   = = = = = = = = = = */

- loadRECPanel:sender {
  if (RECPanel==nil) {
    if (![NXApp loadNibSection: "rec.nib" owner: self withNames: NO])
{return nil;}
    [browser setEmptySelectionEnabled: YES];
    [browser setMultipleSelectionEnabled: NO];
    [browser getTitleFromPreviousColumn: NO];
    [browser setTitle: "components" ofColumn: 0];
    stillUnused=YES;
    }
  [RECPanel orderFront: nil];
  [RECMatrix setEmptySelectionEnabled: YES];
  [RECMatrix selectCellAt: -1: 0];
  [self loadRECSource: RECOButn];
  return self;
  }

/* - - - - - - - - - - - -  c o p y   v i e w s  - - - - - - - - - - - - -*/

- print:sender {[sourceWindow smartPrintPSCode: self]; return self;}

- windowDidBecomeKey:sender {
sourceWindow=sender;
    if (sourceWindow==solutionWindow) sourceView=solutionView;
    if (sourceWindow==phaseWindow)    sourceView=phaseView;
    if (sourceWindow==controlWindow)  sourceView=[controlWindow contentView];
      else sourceView=NULL;
[menuCopy setTarget: self];
[menuCopy setAction: @selector(copy:)];
[menuCopy setEnabled: YES];
[menuPrint setTarget: self];
[menuPrint setAction: @selector(print:)];
[menuPrint setEnabled: YES];
return self;
}

- windowDidResignKey:sender {
[menuCopy setEnabled: NO];
[menuPrint setEnabled: NO];
return self;
}

- copy:sender {
    showRECBack=NO;
    if (sourceWindow==solutionWindow) [self copySomeView: solutionView];
    if (sourceWindow==phaseWindow)  [self copySomeView: phaseView];
    if (sourceWindow==RECPanel) [self copySomeView: [RECPanel contentView]];
    if (sourceWindow==controlWindow)
[self copySomeView: [controlWindow contentView]];
    showRECBack=YES;
return self;
}

- copySomeView:theView {
NXRect zbounds;
id     pBoard;
  showRECBack=NO;
  pBoard = [Pasteboard new];
  [pBoard declareTypes: &NXPostScriptPboardType num: 1 owner: self];
  [theView getBounds: &zbounds];
  [theView writePSCodeInside: &zbounds to: pBoard];
  showRECBack=YES;
return self;
}

/* - - - - S P E C I A L   R U L E  B O X   M A N A G E M E N T - - - - */

- loadRECSource:sender {
  if   ([[sender selectedCell] state])  [self loadLowerPanel: intBox];
  else [self loadLowerPanel: extBox];
return self;
}

- loadLowerPanel:thisBox {
  NXRect bxRect, loRect;
  id cv;
cv=[thisBox contentView];
// printf("number of subviews = %d\n",[[cv subviews] count]);
//  [lilBox setContentView: cv];
  [bigBox setContentView: cv];
//  [bigBox getFrame: &bxRect];
//  [lilBox getFrame: &loRect];
//  loRect.origin.x=bxRect.origin.x+
// (bxRect.size.width-loRect.size.width)/2.0;
//  loRect.origin.y=bxRect.origin.y+
// (bxRect.size.height-loRect.size.height)/2.0;
//  [lilBox setFrame: &loRect];
  [bigBox display];
//  [lilBox display];
return self;
}

/*---------------- B R O W S E R  D E L E G A T E -------------------*/

- (int)browser:sender fillMatrix: matrix inColumn: (int)col {int  i, n;
    n=0;
    printf("browser\n");
     switch(col) {
        case 0:
          for(i=0; i<5; i++) {
            [matrix addRow];
            [[matrix cellAt: i: 0]  setLeaf : NO ];
            [[matrix cellAt: i: 0]  setLoaded : YES ];
            [[matrix cellAt: i: 0]  setEnabled : YES ];
           }
           [[matrix cellAt: 0: 0]  setStringValue: "graph solutions"];
           [[matrix cellAt: 1: 0]  setStringValue: "phase plane"];
           [[matrix cellAt: 2: 0]  setStringValue: "stability diagram"];
   [[matrix cellAt: 3: 0]  setStringValue: "asymptotic values"];
   [[matrix cellAt: 4: 0]  setStringValue: "just definitions"];
           n=5;
           break;
        case 1:
          if(strcmp([[sender selectedCell] stringValue],
 "graph solutions") == 0) {
//          for(i=0; i<NY; i++) {
//            [matrix addRow];
//            [[matrix cellAt: i: 0]  setStringValue: recrule[i]];
//            [[matrix cellAt: i: 0]  setLeaf : YES ];
//            [[matrix cellAt: i: 0]  setLoaded : YES ];
//            [[matrix cellAt: i: 0]  setEnabled : YES ];
//           }
          n=NY;}
          if(strcmp([[sender selectedCell] stringValue],
 "predefined") == 0) {
          for(i=0; i<1; i++) {
            [matrix addRow];
            [[matrix cellAt: i: 0]  setStringValue: idef[i]];
            [[matrix cellAt: i: 0]  setLeaf : YES ];
            [[matrix cellAt: i: 0]  setLoaded : YES ];
            [[matrix cellAt: i: 0]  setEnabled : YES ];
           }
          n=1;}

          if(strcmp([[sender selectedCell] stringValue],
 "just definitions") == 0) {
          for(i=0; i<95; i++) {
            [matrix addRow];
            [[matrix cellAt: i: 0]  setStringValue: dtbl[i].r_cmnt];
            [[matrix cellAt: i: 0]  setLeaf : YES ];
            [[matrix cellAt: i: 0]  setLoaded : YES ];
            [[matrix cellAt: i: 0]  setEnabled : YES ];
           }
          n=95;}

        default: n=0; break; }
      return n;
    }

/* to distract the single click in a browser double click */
- nhul: sender {return self;}

/* - - - - - - -   F I L E   A N D   D I R E C T O R Y   - - - - - - - */

- setFilename:(const char *)aFilename {
  if (filename) free(filename);
  filename = malloc(strlen(aFilename)+1);
  strcpy(filename, aFilename);
  [window setTitleAsFilename:aFilename];
  return self;
}

- load:(const char*)filename {
  int      i, fd;
  NXStream *theStream;
    [self loadRECPanel: self];
    [bigBox setContentView: [extBox contentView]];
    [self setFilename: filename];
    fd = open(filename, O_RDONLY, 06666);
    theStream = NXOpenFile(fd, NX_READONLY);
//      NXScanf(theStream,"%s",cstr);
      i=0;
      while (NXScanf(theStream,"%c",&cstr[i])!=EOF) i++;
      phil=strlen(cstr);
    NXClose(theStream);
    close(fd);
    [RECPanel setTitleAsFilename: filename];
    [RECPanel setDocEdited:NO];
  [execWin setSel: 0: [execWin textLength]];
  [execWin replaceSel: cstr];
  [execWin scrollSelToVisible];
  [execWin display];

return self;
}

- open:sender {
  char *types[3] = {"rec","rec-r",0};
  int  i, fd;
  NXStream *theStream;

  [[OpenPanel new] allowMultipleFiles: NO];
  [[OpenPanel new] chooseDirectories: NO];

  if ([ [OpenPanel new] runModalForTypes: types]) {
    [self setFilename: [[OpenPanel new] filename]];
    fd = open(filename, O_RDONLY, 06666);
    theStream = NXOpenFile(fd, NX_READONLY);
//      NXScanf(theStream,"%s",cstr);
      i=0;
      while (NXScanf(theStream,"%c",&cstr[i])!=EOF) i++;
      phil=strlen(cstr);
    NXClose(theStream);
    close(fd);
    [RECPanel setTitleAsFilename: filename];
    [RECPanel setDocEdited:NO];
  [execWin setSel: 0: [execWin textLength]];
  [execWin replaceSel: cstr];
  [execWin scrollSelToVisible];
  [execWin display];
    }

return self;
}

- saveAs:sender {
  id panel;
  const char *dir;
  char *file;

/* prompt user for the file name and save to that file */

  if (filename==0) {
    /* no filename; set up defaults */
    dir  = NXHomeDirectory();
    file = (char *)[RECPanel title];
    } else {
    file=rindex(filename,'/');
    if (file) {
      dir   = filename;
      *file = 0;
      file++;
      } else {
```

```
      dir  = filename;
      file = (char *)[RECPanel title];
      }
    }
  panel = [SavePanel new];
  [panel setRequiredFileType: "rec-r"];
  if ([panel runModalForDirectory: dir file: file]) {
    [self setFilename: [panel filename]];
    return [self save: sender];
    }
    return nil;  /* didn't save */
  }

- save:sender {
  int      fd;
  NXStream *theStream;

  if (filename==0) return [self saveAs: sender];
  [RECPanel setTitle: "Saving ..."];

  fd = open(filename, O_WRONLY | O_CREAT | O_TRUNC, 06666);
  if (fd < 0) {
    NXRunAlertPanel(0, "Cannot save file: %s",0,0,0,strerror(errno));
    return self;
    }
  theStream = NXOpenFile(fd, NX_WRITEONLY);
  [execWin writeText: theStream];
  NXClose(theStream);
  close(fd);
  [RECPanel setTitleAsFilename: filename];
  [RECPanel setDocEdited:NO];
  return self;
}

- close:sender {const char *fname;
int         q;
  if ([RECPanel isDocEdited])
//    fname=filename ? filename : [sender title];
//    if (rindex(fname,'/')) fname = rindex(fname,'/') + 1;
    q = NXRunAlertPanel(
      "Save",
      "Save changes to %s?",
      "Save",
      "Don't Save",
      "Cancel",
      filename);
    if (q==1) {if (![self save:nil]) return nil;}
    if(q==-1) return nil;
//  [sender setDelegate: nil];
//  [proc free];
//  [self free];
  return self;
}

- windowWillClose:sender {
[self close: self];
printf("windowWillClose\n");
return self;
```

```
}

/* = = = = = = = = =   P r o g r a m   E d i t o r   = = = = = = = = = */

- recMatrix:sender {int i, j;
  i=[sender selectedRow];
  j=[sender selectedCol];
  [comment setStringValue: dtbl[i*32+j].r_cmnt];
return self;
}

- ascii:sender {int i;
  if ([sender selectedRow]==0) i='A'-' '; else i='a'-' ';
  dtbl[i+[sender selectedCol]].xfun();
return self;
}

/* - - - - - - -  R E C   L I N E   C O N F I G U R A T I O N  - - - - */

- run:sender {[self display]; return self;}

- runPhase:sender {
stillUnused=YES;
[phaseWindow makeKeyAndOrderFront: sender];
stillUnused=NO;
[phaseView display];
return self;
}

- runSolution:sender {
stillUnused=YES;
[solutionWindow makeKeyAndOrderFront: sender];
stillUnused=NO;
[solutionView display];
return self;
}

- recompile:sender {
  [execWin getSubstring: cstr start: 0 length: [execWin textLength]];
  [RECPanel setDocEdited:YES];
return self;
}

- useRECLine:sender {
  [RECPanel makeKeyAndOrderFront: sender];
  [RECValue setStringValue: " "];
 return self;
}

- runRECOffLine:sender {
  [RECValue setStringValue: "*"];
  if (rec([sender stringValue])) {
     [RECValue setStringValue: "T"];
     }
  else
    [RECValue setStringValue: "F"];
  return self;
}
```

```
- copyREC:sender {
   if ([sender selectedColumn] == [sender lastColumn]) {
   if(strcmp([sender titleOfColumn: [sender lastColumn]],"graph solutions")
== 0) {
     [RECValue setStringValue: " "];
     [RECLine setStringValue: [[sender selectedCell] stringValue]];
     }}
   return self;
  }

int rec(z) char *z; {if (rec_c(z,rprg,PLEN,dtbl)==1) rec_x(rprg);}

/* = = = = = = = = = = = =     R E C V I E W     = = = = = = = = = = = = = */

- initFrame:(const NXRect *)frameRect
/*
 * Initializes the new RECView object. First, an initFrame: message is sent
 * to super to initialize RECView as a View. Next, the RECView sets its own
 * state -- opaque and that the origin of its coordinate system lies
 * in the center of its area. It then creates and initializes its associated
 * objects, a Storage object, an NXCursor, and a Cell. Finally, it loads into
 * the Window Server some PostScript procedures that it will use in drawing
 * itself.
 */
{
    [super initFrame: frameRect];
    [self translate:floor(frame.size.width/2) :floor(frame.size.height/2)];
    [self setOpaque:YES];
    return self;
}

- drawSelf:(const NXRect *)rects :(int)rectCount
/*
 * Draws the RECView's background and axes.  If there are any points,
 * these are drawn too.
 *
 * (Note:  For simplicity, although RECView only repaints the background
 * of the update region, it redraws the entire axes. For better performance,
 * only those parts of the axes that fall within the update region should
 * be redrawn.)
 */
{
NXPoint *aPoint;

if (rects == NULL) return self;
if (stillUnused)   return self;

/* paint visible area white then draw axes */
PSsetgray(NX_WHITE);
if (showRECBack) NXRectFill(&rects[0]);
PSsetgray(NX_DKGRAY);

printf("drawing self %s\n",cstr);

  isErrm=FALSE;
  [RECValue setStringValue: "*"];
  if (rec(cstr)) [RECValue setStringValue: "t"];
```

```
    else [RECValue setStringValue: "f"];
  if (isErrm) {[comment setStringValue: errm]; isErrm=FALSE;}
  PSstroke();

return self;
}

- sizeTo:(NXCoord)width :(NXCoord)height
/*
 * Ensures that whenever the RECView is resized, the origin of its
 * coordinate system is repositioned to the center of its area.
 */
{
[super sizeTo: width: height];
[self setDrawOrigin: -floor(width/2): -floor(height/2)];
return self;
}

@end

/* end RECView.m */
```

6 Some applications of REC-R

Coefficient matrices from Equation 1 are defined in `sero.c` which is a part of SERO, according to the following extract:

```
/*  [Pot. 1: Onda plana, ec. de Schroedinger] */
void potl1(u,e,z) double *u, e, z; {
u[0]=0.0; u[1]=-2.0*e; u[2]=1.0; u[3]=0.0;}

/*  [Pot. 2: Pozo finito, ec. de Schroedinger] */
void potl2(u,e,z) double *u, e, z; {double h, q;
h=0.5*wi; q=(z<-h||z>h)?0.0:-hi;
u[0]=0.0; u[1]=q-2.0*e; u[2]=1.0; u[3]=0.0;}

/*  [Pot. 3: Barrera finita, ec de Schroedinger] */
void potl3(u,e,z) double *u, e, z; {double h, q;
h=0.5*wi; q=(z<-h||z>h)?0.0:hi;
u[0]=0.0; u[1]=q-2.0*e; u[2]=1.0; u[3]=0.0;}

/*  [Pot. 4: Escalon, ec. de Schroedinger] */
void potl4(u,e,z) double *u, e, z; {
u[0]=0.0; u[1]=((z>=0.0)?st:0.0)-2.0*e; u[2]=1.0; u[3]=0.0;}

/*  [Pot. 5: Oscilador armonico, ec. de Schroedinger] */
void potl5(u,e,z) double *u, e, z; {
u[0]=0.0; u[1]=z*z-2.0*e; u[2]=1.0; u[3]=0.0;}

/*  [Pot. 6: Oscilador cuartico, ec. de Schroedinger] */
void potl6(u,e,z) double *u, e, z; {double x;
u[0]=0.0; x=z*z; u[1] = 0.1*x*x -x -2.0*e; u[2]=1.0; u[3]=0.0;}

/*  [Pot. 7: Oscilador armonico factorizado, ec. de Dirac] */
void potl7(a,e,z) double *a, e, z; {double s, t, u, v;
s=0.1667*z*z-e; t=2.0*s*z; u=em*sin(t); v=em*cos(t);
a[0]=-u; a[1]=v; a[2]=v; a[3]=u;}
```

```
/*  [Pot. 8: Gaussian Potential, ec. de Dirac] */
void potl8(u,e,z) double *u, e, z; {double g;
g=st*exp(-(z*z)/wi); u[0]=0.0;
 u[1]=em+g-e; u[2]=em-g+e; u[3]=0.0;}

/*  [Pot. 9: Sinusoidal potential] */
void potl9(u,e,z) double *u, e, z; {
u[0]=0.0; u[1]=hi*cos(z)-2.0*e; u[2]=1.0; u[3]=0.0;}

/*  [Pot. 10: Linear potential] */
void potl10(u,e,z) double *u, e, z; {
u[0]=0.0; u[1]=-z-2.0*e; u[2]=1.0; u[3]=0.0;}

/*  [Pot. 11: Coulomb potential] */
void potl11(u,e,z) double *u, e, z; {
u[0]=0.0; u[1]=(z>-7.4?-1.0/(z+7.5):0.0)-1.0*e;
 u[2]=1.0; u[3]=0.0;}

/*  [Pot. 12: Inverse Square potential] */
void potl12(u,e,z) double *u, e, z; {
u[0]=0.0; u[1]=((z>=-7.5)?-25.0/((z+7.6)*(z+7.6)):0.0)-1.0*e;
u[2]=1.0; u[3]=0.0;}

/*  [Pot. 13: Damped classical oscillator with forcing] */
void potl13(u,e,t) double *u, e, t; {double g;
u[0]=-0.1; u[1]=-2.0*e; u[2]=1.0; u[3]=0.0;
g=0.2*sin(3.0*t); u[4]=u[5]=g; u[6]=u[7]=0.0;}

/*  [Pot. 14: Dirac Harmonic Oacillator] */
void potl14(u,e,z) double *u, e, z; {double g;
g=0.1666*z*z; u[0]=0.0; u[1]=em+g-e; u[2]=em-g+e; u[3]=0.0;}

/*  [Pot. 15: Factored Inverse Square] */
void potl15(u,e,z) double *u, e, z; {double s, c, g, qq;
qq=0.2*(z+7.6); g=log(0.2*(z+7.6)); s=sin(g); c=cos(g);
u[0]=-(e*c*s*qq); u[1]=-(e*c*c*qq); u[2]=e*s*s; u[3]=e*s*c;}

/*  [Pot. 18: Dirac Sinusoidal potential] */
void potl18(u,e,z) double *u, e, z; {double p;
switch(0) {
case 0:  p=hi*cos(z); break;
case 1:  p=hi*(cos(z)-0.333*cos(3.0*z)); break; /* flat    */
case 2:  p=hi*(cos(z)+0.333*cos(3.0*z)); break; /* sharp   */
case 3:  p=hi*(cos(z)-0.333*cos(3.0*z)+0.2*cos(5.0*z));
                                         break; /* flatter */
case 4:  p=0.67*hi*(cos(z)+0.333*cos(3.0*z)+0.2*cos(5.0*z));
                                         break; /* sharper */
case 5:  p=0.2*hi*(cos(z)+cos(2.0*z)+cos(3.0*z))+
                  cos(4.0*z)+cos(5.0*z); break; /* deltoid */
default: p=hi*cos(z); break;
}
u[0]=0.0; u[1]=em+p-e; u[2]=em-p+e; u[3]=0.0;}

/*  [Pot. 19: Quarkish Harmonic Oscillator] */
void potl19(u,e,z) double *u, e, z; {
u[0]=hi*z; u[1]=em-e; u[2]=em+e; u[3]=-hi*z;}
```

The results would be much more attractive if they were displayed in a table rather than just as C subroutines; some of them are shown below.

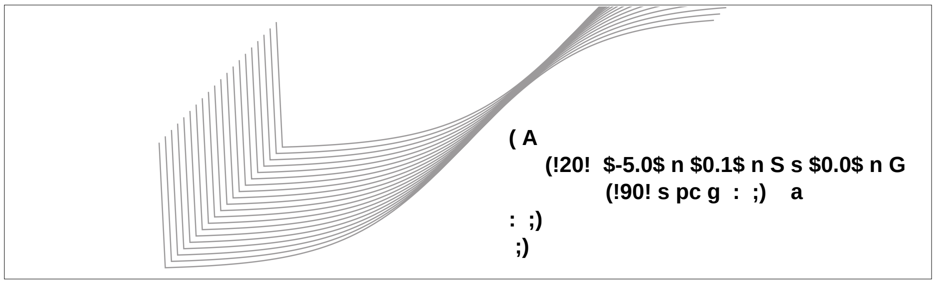
index	coefficient matrix	comment
1	$\begin{bmatrix} 0 & -2e \\ 1 & 0 \end{bmatrix}$	V1 : plane wave
5	$\begin{bmatrix} 0 & z^2-2e \\ 1 & 0 \end{bmatrix}$	V5 : Schrödinger harmonic oscillator
6	$\begin{bmatrix} 0 & \frac{1}{10}x^4-x^2-2e \\ 1 & 0 \end{bmatrix}$	V6 : Schrödinger quartic oscillator
7	$\begin{bmatrix} -m\sin(\frac{1}{3}x^3-2ex) & m\cos(\frac{1}{3}x^3-2ex) \\ m\cos(\frac{1}{3}x^3-2ex) & m\sin(\frac{1}{3}x^3-2ex) \end{bmatrix}$	V7 : factored Dirac oscillator
8	$\begin{bmatrix} 0 & m+he^{-x^2/w}-e \\ m-he^{-x^2/w}+e & 0 \end{bmatrix}$	V8 : Gaussian potential, Dirac eqn.
9	$\begin{bmatrix} 0 & h\cos(x)-2e \\ 1 & 0 \end{bmatrix}$	V9 : cosine : Mathieu equation
10	$\begin{bmatrix} 0 & -x-2e \\ 1 & 0 \end{bmatrix}$	VA : linear potential : Airy function
14	$\begin{bmatrix} 0 & m+\frac{1}{6}x^2-e \\ m-\frac{1}{6}x^2+e & 0 \end{bmatrix}$	VE : Dirac harmonic oscillator
18	$\begin{bmatrix} 0 & em+h\cos(x)-e \\ em-h\cos(x)+e & 0 \end{bmatrix}$	VI : Dirac sinusoidal
19	$\begin{bmatrix} hx & m-e \\ m+e & -hx \end{bmatrix}$	VJ : quarkish oscillator

The first example, shown in Figure 9, solves no differential equation, but it illustrates the machinery for which REC is considered as a useful interface to some other program. Of course, even such a small example is still too complicated to be typed casually at the console, but its program-like structure incorporates many choices and initializations that would be harder to accomodate in the graphical user interface style.

The inner loop, whose purpose is to run out a single graph whose parameters remain constant, consists of the REC expression `(!90!  s pc g :  ;)` whose counter calls for 90 points on the graph. The graphic variable `s` is autoincrementing and goes onto the pushdown list first as the x-coordinate.

Since it autoincrements, it cannot be used a second time to get the y-coordinate, but its value can be duplicated by pushing, vololowing which it is clipped by a hyperbolic tangent which leaves a fairly true interval $-1 < y < 1$, but quickly flattens the variable elsewhere.

The next outward loop has charge of drawing a family of coves, in this case 20. It is the place where a parameter governing the curve could be varied, but that is missing in this example. But the tail of the loop *does* contain the operator `a`, responsible for the increasing offset as each curve is drawn. Correspondingly, the head of this loop initializes the x coordinate anew (and independently of the action of the offset, which is only applied as the graph is being drawn).This is done by the sequence -5.0 `n` 0.1 `n S` which feeds the origin -5.0 and increment 0.1 to the initializing operator `S`.



```
( A
     (!20!  $-5.0$ n $0.1$ n S s $0.0$ n G
            (!90! s pc g  :  ;)    a
: ;)
 ;)
```

Figure 9: A rudimentary REC-R program which only graphs a constant, but clipped, straight line. It illustrates the initialization of the graphing variable, the application of offsets to successive graphs, and the advantages of clipping, to keep the entire body of the graph within the graphing area. Hidden line suppression would not alter this graph, but there is a noticable jump where the first value has not been chosen carefully. Sometimes this "bug" is a "feature" as when it shows a zero-point.

The choice of these numbers is governed by the fact that the screen is comparable in size to a sheet of paper, so that ten inches measures the extent of the graphing area, in which the pixels are maybe hundredths of an inch, so ten pixels, or a tenth of an inch, is a good increment. Actually, *PostScript* uses points, and actually, SERO scales everything anyway, so the numbers cited are the ones to think about when planning the dimensions of graphs.

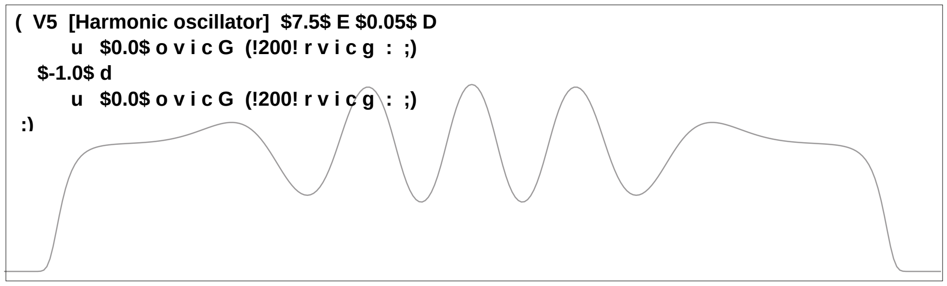
The initial point for graph movements is established by `s` 0.0 `n G`, which shows why all the curves drop down to zero at the left. That is deliberate, but it could be avoided by writing `s p G` instead.

Finally, the offset operating in the middle loop has to be initialized in the outer program by the operator `A`.

Going on to a more complicated example with a potential and for which the graphing is driven by the Runge-Kutta variable, Figure 10 shows the solution at a single energy for potential number 5, the Schrödinger harmonic oscillator.

This time there is a preface `V5 [Harmonic oscillator]` 7.5 `E` 0.05 `D` which selects the potential, sets the energy level, and selects a Runge-Kutta step size, which is half the default. It is followed by the first inner loop, which initializes the differential equation, sets up the first point in the graph, and then runs out 200 points and graphs them, namely `u` 0.0 `o v i c G (!200! r v i c g : ;) .`

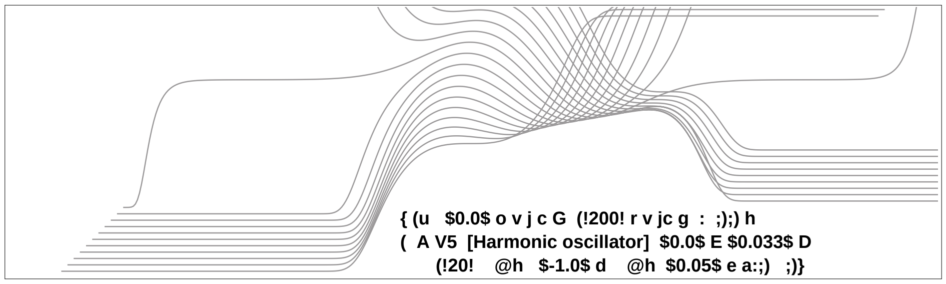
The operator `i` means that the derivative of the odd solution, which is even, is being graphed. To get the solution itself, `j` should have been chosen. There is no reason they could not both have been graphed in the same box, if the additional code had been included. That is far preferable to trying to graph both solutions at once out of the same matrix, but for more extensive calculations it would have to be a trade-off between the costs of Runge-Kutta steps and pen movements.



```
( V5  [Harmonic oscillator]  $7.5$ E $0.05$ D
      u  $0.0$ o v i c G  (!200! r v i c g  :  ;)
   $-1.0$ d
      u  $0.0$ o v i c G  (!200! r v i c g  :  ;)
 :)
```

Figure 10: Another rudimentary REC-R program which graphs a single wave function at one energy of the Schrödinger harmonic oscillator. Outside the potential well the function is exponential, but it has been clipped for aesthetic reasons, namely keeping the graph within a bounding box.

The line between the two inner loops reverses the direction of the Runge-Kutta step.



```
{ (u   $0.0$ o v j c G  (!200! r v jc g  :  ;);) h
( A V5  [Harmonic oscillator]  $0.0$ E $0.033$ D
     (!20!    @h   $-1.0$ d    @h  $0.05$ e a:;)   ;)}
```

Figure 11: A more ambitious version of the program which graphs a single wave function shows twenty wave functions at small increments, crossing the value at which the asymptotic wave function changes sign.

In Figure 11 a second loop has been added which varies the energy eigenvalue. The result is graphed plain, without the suppression of hidden lines because the result is still intelligible, and in fact, slightly preferable in that form. The interesting aspect of the graph is that the energy crosses the value at which the sign of the asymptotic exponential changes. Assuming, by continuity, that there is an energy at which the asymptotic value is zero, one has found one of the quantized energy levels.

Because subroutines are used in the program of Figure 11, they are named and enclosed along with the main program between a pair of braces. There is no way for a main program to call itself, although that is a recursive possibility. Should such a thing be required, it is easy enough to assign the main program some name, create a new main program whose only content is to call the old main program, and procede as though nothing had happened. Who is it that once said that it is a poor program which does not even know its own name?

Figure 12 shows a more elaborate construction, in which the predicate sensing the difference between the classically accessible region and the classically forbidden region has been used to color the two parts of the graph differently.

It is a commentary on the mechanism maintaining the horizons that this figure cannot be rendered with hidden lines suppressed. The reason is that drawing the two halves of each curve separately, outwards from the center, runs counter to the indexing scheme for the horizon. This is a blemish which could, of course, be fixed.

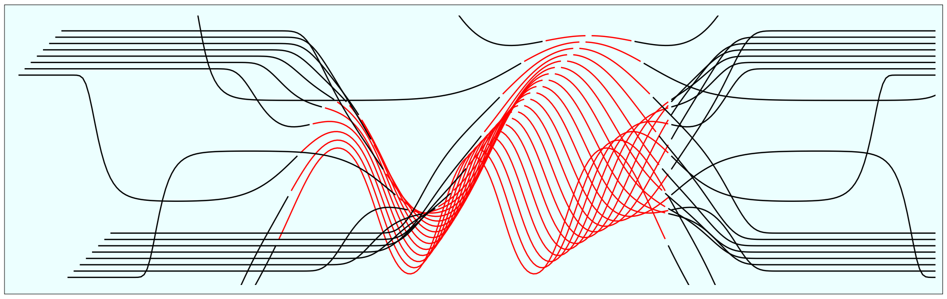


Figure 12: A more ambitios version of the program which graphs a single wave function shows twenty wave functions at small increments, crossing the value at which the asymptotic wave function changes sign.

The program which drew Figure 12 is the following:

```
{
([ put 150 increments per trajectory] !150! ;) n
([ solution 10 in classical region]  Cr  v l c  G  (r  F;  @n v l c g : ); ) i
([ solution 10 in forbidden region]  Ck v l c  G  (r  (F); @n v l c g : ); ) I
([ get the solution in one direction]  u $0.0$ o  (r  @i @I: ;) ;) h
    (  A V5  [Harmonic oscillator]   $5.0$ E $0.05$ D $-1.0$ U
       (!20!  (@n:;)  @h   $-1.0$ d   (@n:;)  @h  $-0.25$ e a: ;)   ;)
}
```

The counter @n has been set aside as a subroutine to maintain the continuity of counting in either of the two regions of the graph, however they alternate. Placing a counter in a subroutine is a standard technique to access the same count from different parts of a program; of course it must be initialized, as is done in (@n:;), by exausting the count whenever it needs to be reset.

There are two predicates in REC-R which can be used for coloring surfaces, b which tests the sign of the variable on the pushdown list, and B which tests whether the variable lies in the range $-1 < y < 1$. The latter is mostly used in judging whether angles of rotation are real or imaginary, often used in constructing stability charts for periodic equations.

Although REC-R has a pushdown list, there is no provision for using it like a hand-held calculator. Some future version may consider the extension useful, at the very least for forming linear combinations of matrix alememnts or of solution vectors.

The examples which have been shown up to now utilize the fixed rectangular "solution window" in REC-R. To use this window, which is always centered on zero, it is convenient to keep in mind that ± 10.0 is the range of the independent variable and ± 2.0 of the dependent variable, more or less. A reference line representing the x-axis could be generated by the REC sequence ([reference axis] A -7.5 n 0.0 n G 7.7 n 0.0 n g;), but any other line could be made by choosing suitable coordinates.

The other window, the "phase window," follows the same scaling, but its overall size and shape are adjustable by the sizing bar on its bottom margin. To use this window, it is only necessary to press the corresponding button and, of course, to have chosen a suitable pair of variables for the coordinates.

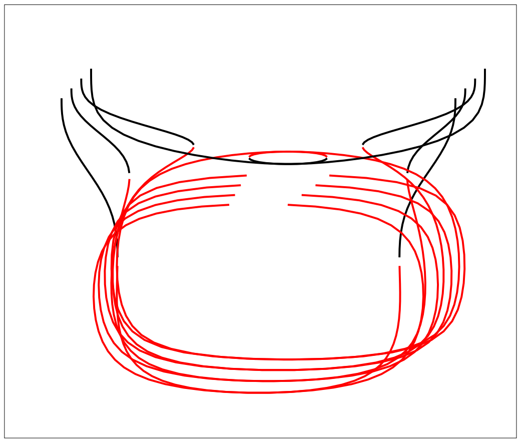


Figure 13: A few phase plane trajectories, with offsets, are graphed to compare their behavior near a resonance.

Figure 13 exhibits such a window with the harmonic oscillator phase plane. The corresponding code, which uses `j` and `l` to summon the coordinates rather than `l` and `v`, is:

```
{ ([ put 100 increments per trajectory] !100! ;) n
([ solution 10 in classical region] Cr j c l c  G (r  F;  @n j c l c g : ); ) i
([ solution 10 in forbidden region] Ck j c l c  G (r (F); @n j c l c g : ); ) I
([get the solution in one direction]  u $0.0$ o  (r  @i @I: ;) ;) h
    ( A V5  [Harmonic oscillator]   $3.0$ E $0.05$ D $-0.75$ U
      (!4!  (@n:;)  @h  $-1.0$ d  (@n:;)  @h  $0.4255$ e a: ;) ;) }
```

Of course, neither window has an exclusive use, and having two is probably redundant. When a large graph is desired, consisting of a large number of offset images, the phase window is especially convenient.

References

[1] Harold V. McIntosh and Gerardo Cisneros, "The programming languages REC and Convert," *SIGPLAN Notices*, **25** 81-94 (July 1990).

[2] Gerardo Cisneros, "Configurable REC," *ACM SIGPLAN Notices* **29** May 1995 (11 pages)

[3] Simson L. Garfinkel and Michael K. Mahoney, *NeXTSTEP^{TM} Programming*, Springer Verlag, New York, 1953. ISBN 0-387-97884-4

[4] Adobe Systems, Incorporated, *PostScript Language Reference Manual*, Second Edition, Addison-Wesley Publishing Company, Inc., Reading, Massachussetts, 1990. ISBN 0-201-18127-4.

[5] Earl A. Coddington and Norman Levinson, *Theory of Ordinary Differential Equations*, McGraw-Hill Book Company, New York, 1955.

[6] Philip Hartman, *Ordinary Differential Equations*, John Wiley & Sons, Inc, New York, 1964.

January 11, 2001