dstool:
A Dynamical System Toolkit with an Interactive Graphical Interface

Based on the program **kaos** written by S. Kim and J. Guckenheimer

Reference Manual

J. Guckenheimer
M. R. Myers F. J. Wicklin P. A. Worfolk

Center For Applied Mathematics
Cornell University, Ithaca NY 14853

September 1991 Draft - Updated May 1995

Contents

1 Saving and Loading Data 4
- 1.1 DsTool Format . . . 4
  - 1.1.1 Settings . . . 4
  - 1.1.2 Configuration . . . 6
  - 1.1.3 Memory Objects . . . 7
  - 1.1.4 Auxiliary Function Values . . . 9
- 1.2 Loading Unformatted Data . . . 10

2 Printing Data 13
- 2.1 Coordinate System . . . 13
- 2.2 DsTool Format . . . 14
- 2.3 Piping Data to Printers . . . 14
- 2.4 Customizing the Print Environment . . . 15

3 Creating Custom User Applications 17
- 3.1 The DsTool Interface . . . 17
  - 3.1.1 Writing the Required Routines . . . 17
  - 3.1.2 Modifying user_panels.c . . . 19
  - 3.1.3 Updating the User Library Makefile . . . 20
- 3.2 Creating a Custom Window . . . 20
  - 3.2.1 The Example . . . 20
  - 3.2.2 Designing the Window with GUIDE . . . 21
  - 3.2.3 Customizing the Window Creation Code . . . 21
  - 3.2.4 Writing the Panel Handlers . . . 24
  - 3.2.5 Writing the Standard Panel Procs . . . 26
  - 3.2.6 Adding a New Library . . . 33

4 Adding User Transformations to the Multiple Panel 35
  - 4.0.7 Writing the Transformation . . . 35
  - 4.0.8 Declaring the Transformation . . . 37
  - 4.0.9 Making the Transformation Available . . . 37

5 Data Storage **38**
5.1 The Memory Object 38
5.1.1 Creating and Destroying Memory Objects 39
5.1.2 Writing to Memory Objects 40
5.1.3 Reading Memory Objects 42
5.1.4 Other Memory Object Procedures 45

List of Tables

1.1 Integer codes for plotting symbols within DsTool. 12

Chapter 1

Saving and Loading Data

For portability and convenience, we have chosen to perform all I/O file operations using ASCII text files. For large data, this choice implies that reading and writing files will be slower than if the data were written in binary format.

1.1 DsTool Format

When DsTool saves data, it first writes a short header consisting of the user's name, the date, and the dynamical system being studied. Depending upon which save options have been selected, DsTool then writes an appropriate block of data. There are four categories of data which DsTool may save: settings, configuration, memory objects, and auxillary function values. Each of this categories will be discussed in turn.

The format of DsTool Version 2.0 files has been changed from Version 1.x. This is essentialy due to the nature of the new postmaster. The postmaster can now write a file which reflects much of its status. This file is written in tcl format and may be executed by tcl in order to reset the postmaster as much as possible to the state it was in when the file was saved. This file can be read directly by DsTool or passed through the tcl interpreter. Additional tcl commands may be embedded in this file, but then it must be read through the tcl interpreter.

One consequence of this is that Version 2.0 DsTool data files are an excellent guide to which postmaster fields exist.

1.1.1 Settings

The *setting* option on the Save Panel is used to record values on all panels which record their data in the postmaster. For Version 2.0, this feature is automatic and includes many contributed panels. In Version 1.x, only a subset of panels explicitly supported by save code was covered. This still included enough fields to establish the basic state of DsTool.

If a user wants to save the current state of DsTool (that is, current parameters, initial conditions, colormaps, sizes of windows, etc.) then the user should simultaneously save both settings and the DsTool window configuration in a file. When the user later reloads

this data file into DsTool, DsTool will be in a nearly identical state as it was when the data file was written.

The following data block in Version 1.x describes a few settings for the Lorenz system:

```
# System_Name  Lorenz System
# Varb_Dim     4
# Param_Dim    3
# Varb_Ic      0.1 0.1 0.1 0
# Varb_Fc      0 0 0 0
# Varb_Min     -30 -30 5 0
# Varb_Max     30 30 50 100
# Param        10 28 2.66666666
# Param_Min    -20 0 -5
# Param_Max    20 40 5
# Function_Dim 2
# Function_Min -60 -60
# Function_Max 60 60
...
```

The corresponding version in DsTool 2.0 format is:

```
pm PUT Load.Model_Name "Lorenz System"
pm EXEC Load.If_New_Model_Fnct

pm PUT_LIST Defaults.Varb_Max 0 3 30 30 50 100
pm PUT_LIST Defaults.Varb_Min 0 3 -30 -30 -5 0

pm PUT_LIST Selected.Param_Ic 0 2 10 28 2.6666667
pm PUT_LIST Selected.Varb_Fc 0 3 0.1 0.1 0.1 0
pm PUT_LIST Selected.Varb_Ic 0 3 0.1 0.1 0.1 0

pm PUT_LIST Defaults.Funct_Max 0 1 60 60
pm PUT_LIST Defaults.Funct_Min 0 1 -60 -60

...
```

Several differences are evident. The Version 2.0 format is more verbose. As dividends, though, the format is readily interpretable by a tcl based interpreter and is very close to the internal syntax of postmaster calls in the code. Another new feature are the "pm EXEC *function_name* ..." lines. These tell the system to execute a function *function_name* with no arguments. (Information that might have been supplied by arguments is instead retrieved by explicit postmaster calls in *function_name*).

Also certain fields are no longer saved in Version 2.0. For example the dimension of the extended phase space is set in the model description file. While the loading of a model does record this information in the postmaster "Model.Varb_Dim" field, this field cannot be safely altered without loading a new model. Hence it is not saved.

The remainder of this section describes a number of the Version 1.x keywords. Functionality of the corresponding "pm" constructions in Version 2.0 is similar.

The keyword `System_Name` will cause DsTool to examine the name of the currently loaded dynamical system. If the current name matches the title string which follows the

command (in this case, "Lorenz System") then no action is performed. If, however, the title string does not match the name of the current system, then DsTool will attempt to locate an installed system whose name does match. If a match is made, then DsTool will change models; otherwise it will attempt to load settings into the current model.

The keywords `Varb_Dim` and `Param_Dim` tell DsTool the dimensions of phase space and parameter space of the data. If the currently loaded dynamical system has a phase and/or parameter space whose dimensions do not agree with the data dimensions, then the user is presented with the option of cancelling the loading operation. It is up to the user to insure that the data from one dynamical system is not inadvertently loaded into the memory of a second dynamical system.

The keywords `Varb_Ic` and `Varb_Fc` set the initial and final condition for the phase space variable; similarly, `Param` sets the current parameter values. The keywords `Varb_Min` and `Varb_Max` set the default range for phase space variables, and similarly `Param_Min, Param_Max, Function_Min`, and `Function_Max` set the default ranges for parameters and function values. The keyword `Function_Dim` notifies DsTool as to how many function it should expect. Because functions may be frequently added or removed from a dynamical system as needs change, we do not present an error message if the dimension given by `Function_Dim` is different than the current number of installed functions. We handle this discrepancy internally.

Most of the remaining settings which may be saved are used to compute orbits, find fixed points, and compute stable and unstable manifolds of fixed points.

1.1.2 Configuration

The *configuration* refers to the size and placement of DsTool windows. The Version 2.0 format for this information is of the form:

```
pm PUT Win.Current Orbits
pm PUT_LIST Win.Locn.Orbits 0 3 699 12 316 370
pm EXEC Win.Open_Current

pm PUT Win.Current Selected
pm PUT_LIST Win.Locn.Selected 0 3 51 167 319 252
pm EXEC Win.Open_Current

...

pm PUT TwoD_Win.Window_Number 0
pm PUT_LIST TwoD_Win.Locn 0 3 36 364 534 502
pm EXEC TwoD_Win.Open_Corresponding

pm PUT TwoD_Win.Symbol_Index 0
pm PUT TwoD_Win.Sym_Size_Index 1
pm PUT TwoD_Win.Bg_Color_Index 0
pm PUT TwoD_Win.Show_Cmap 0
pm PUT TwoD_Win.Cmap_Type_Index 0
pm PUT TwoD_Win.Depth_Coord 0
pm PUT TwoD_Win.Depth_Coord_Min -30
```

```
pm PUT TwoD_Win.Depth_Coord_Max 30
pm PUT TwoD_Win.Cmap_File colortable_alt
pm PUT TwoD_Win.Hor_Min -30
pm PUT TwoD_Win.Hor_Max 30
pm PUT TwoD_Win.Vert_Min -30
pm PUT TwoD_Win.Vert_Max 30
pm PUT TwoD_Win.Hor_Type 100
pm PUT TwoD_Win.Ver_Type 100
pm PUT TwoD_Win.Hor_Index 0
pm PUT TwoD_Win.Ver_Index 1
pm EXEC TwoD_Win.Update_Current
```

The last four integers in the "pm PUT_LIST Win.Locn.Orbits 0 3 699 12 316 370" line record the screen locations of the window's left edge, top, width, and height. The first two integers "0 3" are part of the postmaster PUT_LIST syntax and indicate that positions 0 thru 3 of the list are being filled.

In Version 1.x, you can save the size and location of the following windows and panels: the Command Window, the Orbits Panel, the Multiple Orbits Panel, the Fixed Point Panel, the Selected Point Window, the Function Window, and the Defaults Settings Window. In addition, each view window may be saved, along with all of the options associated with it.

The configuration in Version 1.x is accomplished by saving command lines such as

```
# Configuration Orbit_Win 143 124 316 369
# Configuration Selected_Win 455 124 275 252
# Configuration Defaults_Win 819 372 347 451
# Configuration Cmd_Win 5 7 435 81
# Configuration TwoD_Win 291 372 534 526
        Window_Number 0   Symbol    0   Size 0           BG_Color 0
        Show_Cmap     0   Cmap_Type 0   Depth_Coord 0    DC_Min -30    DC_Max 30
        Color_Table   colortable_alt
        Hor_Min -30   Hor_Max 30    Ver_Min -30   Ver_Max 30
        Hor_Type 100  Ver_Type 100  Hor_Index 0  Ver_Index 1
```

Each window name (`Orbit_Win, Selected_Win`, etc.) is followed by four integers which signify the screen locations of the window's left edge, top, width, and height. The view window (here a 2D-view window) is followed by information which determines how the window will display data. Much of this information (e.g., symbol and symbol size) is found on the View Options Panel associated with the View Window.

1.1.3 Memory Objects

Internally, DsTool stores data as a *memory object*. There are currently six types of memory objects: trajectory data, multiple trajectory objects, fixed point data, continuation data, parameter data, and selected point data. Parameter data differs from selected point data in that parameter data is displayed in those view windows whose coordinate axes are all parameters; selected point data is not displayed in those view windows, but is displayed in all others. Each memory object consists of a *header* and a *body*. The header contains information which is relevant to all elements of the memory object; the body consists of a list of points. See Section 5.1 for more information about the structure of memory objects.

In Version 2.0, the body of a memory object is stored as a list of lists of lists. This comes from the data structure organization of a point as a list of reals, a trajectory as a list of points, and a flow as a list of trajectories.

A typical Version 2.0 saved memory object might be:

```
pm PUT Cur_Memory.Mem_Type Mult
pm PUT Cur_Memory.Num_Objects 2
pm PUT Cur_Memory.Num_Header_Doubles 0
pm PUT Cur_Memory.Num_Header_Integers 0
pm PUT Cur_Memory.Num_Body_Doubles 7
pm PUT Cur_Memory.Num_Body_Integers 3
SET new_object {
{
{ 14.9172413793 12.3622047244 0.1 0 10 28 2.66666666  }
{ 2 0 2  }
{ 16.5610945055 25.5517463211 11.5804621863 0.04 10 28 2.66666666  }
{ 2 0 2  }
{ 17.5267359415 27.729271084 15.7650797361 0.05 10 28 2.66666666  }
{ 2 0 2  }
}
{
{ 16.1172413793 12.3622047244 0.1 0 10 28 2.66666666  }
{ 2 0 2  }
{ 17.5179627246 26.1914400011 12.6298322542 0.04 10 28 2.66666666  }
{ 2 0 2  }
{ 18.4500045603 28.2798007755 17.1416421165 0.05 10 28 2.66666666  }
{ 2 0 2  }
}
}
```

Such a memory object could be generated by the mult panel using two trajectories each consisting of 3 points. The "{ 2 0 2 }" integer entries are color (first alt_color, then pick_color) and symbol data for each point.

In Version 1.x, when memory objects are stored, the header information is stored first, followed by the body. The following is an example of a memory object for trajectory data:

```
# New_Traj
# Objects 1
# Double_Params 3 : 10 28 2.66666666
# Integer_Params 3 : 0 0 1
# Doubles= 4 Integers= 0
# New_Obj:  500 Points
3.33333333333 5.11111111111 0.1            0.0
3.54567210552 6.01044205838 0.285917212245 0.01
... (total of 500 points) ...
5.07133527463 10.2650449956 1.57313739354  5.00
```

A block of data such as the above will begin with one of the six strings `New_Traj`, `New_Mult`, `New_Fxpt`, `New_Cont`, `New_Param`, `New_Sel_Pt` which identify the type of data which follows. The above data defines a trajectory for a vector field which has three parameters and four phase space variables—three spatial variables plus time. The parameters

are stored in the header whereas the phase space variables are stored in the body. In the above data, three integer parameters in the header are used to indicate the symbol type, size, and color of the trajectory.

Note that in DsTool the independent variable (often time) is given full status as a member of phase space. Consequently, the "DsTool dimension" of phase space may be greater than the commonly accepted dimension. For example, vector fields which are time independent have the same "DsTool dimension" as vector fields which are time-dependent. This has been done so that DsTool may handle autonomous and non-autonomous systems in a uniform way. It should not be a source of confusion unless you intend to load data into DsTool which has been written by an external application. See section 1.2 below.

1.1.4 Auxiliary Function Values

The values of auxillary functions do not comprise a memory object; instead, the function values are recomputed every time they are needed. As a result, DsTool does not provide a way for function values to be loaded from a data file. The user may, however, write these values to a file in the event that he or she wishes to manipulate them using some other software package.

When DsTool writes function values to a file, it loops over every memory object with the exception of parameter data. The body of the memory object is printed and then all of the auxillary functions at each point. In Version 2.0, the outcome is of the form:

```
# Varb_Dim 4
# Param_Dim 3
# Function_Dim 2
SET func_object {
{
{3.33333333 5.11111111 0.1        0.0  10.0 28.0 2.66666666 8.44444444 -1.77777777 }
{3.54567210 6.01044205 0.28591721 0.01 10.0 28.0 2.66666666 9.55611416 -2.46476995}
...
```

In Version 1.x, a typical data block of function evaluations for trajectory data looks like:

```
# New_Func for memory object: Trajectory
# Varb_Dim 4
# Param_Dim 3
# Function_Dim 2
3.33333333 5.11111111 0.1        0.0  10.0 28.0 2.66666666 8.44444444 -1.77777777
3.54567210 6.01044205 0.28591721 0.01 10.0 28.0 2.66666666 9.55611416 -2.46476995
...
... total of 500 points of the form:
... x0         x1         x2      x3   p0   p1      p2      f0(x,p)     f1(x,p)
...
5.07133527 10.2650449 1.57313739 5.00 10.0 28.0 2.66666666 15.3363802 -5.19370972
```

Previously we gave an example of a memory object for trajectory data. This is the corresponding entry for the function values associated with that memory object. Each line of this data block begins with values of the phase space variables (for this example, the first four double-precision numbers) and the parameters (the next three doubles), that is, with

a point x in phase space and a point p in parameter space. Each auxillary function is then evaluated at (x, p) and the resulting scalar quantity is written. In the above example, there are two functions: $f_0(x, p) = x_0 + x_1$ and $f_1(x, p) = x_0 - x_1$.

1.2 Loading Unformatted Data

It is probable that users of DsTool will occasionally generate data outside of DsTool (via another software program). If desired, it is possible to load this data into DsTool in order to view, print, or manipulate it. This can be done by selecting the Unformatted option from the Load Window. Doing so causes the load window to extend vertically, thereby revealing additional panel items which are used to control the loading of unformatted data. Refer to the User's Manual for more information about the Load Window and the meaning of each panel item.

An unformatted data file consists of an optional header, followed by a list of floating point numbers and integers. In many cases these numbers will be arranged in columns. Each line of the header must begin with the comment symbol # (the pound sign). As soon as DsTool reads a line which does not begin with a pound sign, it will assume that it should begin to read data. A typical header looks something like the following:

```
# Written by Tom Jones, Sun Jun 16 19:20:43 EDT 1991
# This file contains fixed points for a time-indep 4D symplectic mapping.
# They were calculated using Maple.
# The parameters for this run are:  a = 0.3; b = -0.734; c = 1.24;
# These fixed points occur on the plane  w = 0.
# The variables contained here are x,y,z.
# Color is ON
# Symbols are ON
```

It is suggested that every data file have a header of the above type which identifies what the file contains and what relevant system parameters were at the time that the file was created.

When loading unformatted data, the user is given the option to load the data into an appropriate memory object. When the user chooses the memory object which will contain the unformatted data, DsTool loads the data. *It is assumed that the unformatted data is appropriate for the chosen data object.* Furthermore, each file may contain at most one type of data. For example, if a data file contains trajectory data, then that same data cannot also contain fixed point data.

When loading unformatted data, DsTool always reads the current values of the phase space variables, the parameters (visible in the Selected Point Panel), the plotting symbol and size (visible in the Defaults Panel), and the current plotting color. Unless these values are set by the unformatted data file being read, they are used as default values. For example, if the user is reading in fixed point data and does not set color or symbol information within the file, then the current color and symbol are used to plot the unformatted data. In this same example, if the user does not include information on parameter values, then the current parameters are used.

We've already looked at a sample header for an unformatted data file. The remainder of that same file might look something like the following: