# DsTool: A Dynamical System Toolkit with an Interactive Graphical Interface

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

User's Manual

J. Guckenheimer

B. A. Meloon M. R. Myers F. J. Wicklin P. A. Worfolk

Center For Applied Mathematics
Cornell University, Ithaca NY 14853

Version Tk Draft - December, 1997

# Contents

# List of Figures

# Chapter 1

# Introduction

## 1.1 DsTool and Dynamical System Toolkits

A *dynamical system* is defined by a set of rules or transformations for determining how points in a multi-dimensional space move in time. Time may either be discrete or continuous. The traces of the points as they move in discrete or continuous time are called *trajectories*. The goal of dynamical systems theory is to provide a comprehensive description of the geometric structures arising from these trajectories. In addition to elucidating the dynamics associated to an individual dynamical system, bifurcation theory may be used to describe how the dynamics of a system varies with changes in parameter values.

Interactive numerical and graphical exploration are important tools in dynamical systems research for several reasons:

- There is generally no way to obtain trajectory information other than by iteration or numerical integration;
- The geometric structures of a dynamical system are often intricate and extremely sensitive to changes in the system parameters, so that interactive computation and graphics have proved useful in interpreting their meaning;
- Exploration of a dynamical system often involves the generation of large quantities of data. Usually, only a small proportion of data must be saved, and typical computer facilities seldom allow for the indiscriminate storage of information. Therefore, it is important to find compact and efficient representations of a system's dynamics that can be easily retrieved.

Consequently, there is a critical need for computational environments that provide effective tools for exploring dynamical systems with minimal effort on the part of the user. Research that relies upon the investigation of dynamical systems would be greatly enhanced by a standard, uniform environment for the exploration of these systems with computers.

The explosion of the graphical computational capabilities of relatively inexpensive desktop computers in the past few years makes the development of such an environment both feasible and timely. This document describes an implementation of one such environment for SunOS, Solaris, SGI, and Linux platforms. The *toolkit* that we describe is an efficient research tool that integrates a graphical user interface, data management capabilities, and a rich set of numerical algorithms together with the flexibility to add more algorithms and communicate data with other programs. The program, called *DsTool* (pronounced dee-ess-TOOL), has been implemented for use with the X Window system from MIT. This version uses the Tcl/Tk package by John Ousterhout for its graphical interface. In addition, Geomview is used for three dimensional graphics and animation capabilities. DsTool is based upon the program **kaos**, written by S. Kim and J. Guckenheimer.

# Chapter 2

# Attributes of Interface Windows

## 2.1 Conventions

The original version of the DsTool toolkit was written to conform to the SUN *Open Look*© conventions. This version was written using the Tcl/Tk toolkit, but adheres to the Open Look conventions whenever possible. For example, a menu entry with trailing ellipses implies the entry will generate a pop-up window if selected. We will also use the Open Look nomenclature SELECT, ADJUST, and MENU when referencing the mouse buttons. (If your mouse has three buttons then these are usually the left, middle and right buttons, respectively.)

Many of the functions of DsTool can be executed by using the keyboard instead of the mouse. In general, pressing `Tab` while the cursor points into a window will cause a panel item to be highlighted with a black outline. Pressing `Tab` again will cause the outline to switch to the next exclusive or non-exclusive setting, listbox, scrollbar, button, or read-write text field. (See below for descriptions of these panel items.) Pressing `Return` or `Space` in one of these items will have different effects depending upon the type of button. The following conventions and notation are used throughout:

**Window title:** The text located in the window header.

**Function:** A brief summary of the window's main purpose within DsTool.

**Description:** Instructions on how to open the window and a detailed account of the window's behavior when the user interacts with it.

**Panel items:** A complete listing of all items located on the window's control panel, along with a brief description of each item's characteristics. The terms used in this document are:

- Button: There are five types of buttons used in DsTool. The first type of button is the *command* button, which initiates an action when selected. Some actions are potentially dangerous (*e.g.*, writing data to a file which already exists) and such actions are accompanied by a *notifier* which requests confirmation. A command button has the command it will execute designated on its face. The second type of button is the *window* button which opens a pop-up window when selected. A window button has the name of the pop-up window it will open followed by ellipsis (...) marks designated on its face. A third type of button is the *command-window* button, which when selected first initiates an action, and then opens a pop-up window. A command-window button has the command it will execute followed by ellipsis marks designated on its face. The fourth type of button is the *menu* button, which displays a menu of options when SELECT is pressed while the pointer is on it. A menu button is a flat button whose label has its first letter underlined. The fifth type of button is the *popup menu* button, which functions just like a menu button. A popup menu button is a raised button with a thin rectangle at the far right of its face. Popup menu buttons are usually used for stack settings. Of these types of buttons, only command, command-window, and window buttons respond to keyboard input. Pressing `Space` in one of these buttons is equivalent to pressing the SELECT button. Pressing `Return` does nothing.

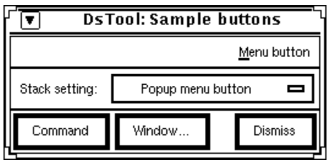


Figure 2.1: Five types of buttons.

- Text Fields: A text field is an area where numerical or alphabetic information may be stored and displayed. Most text fields are *read-write*, meaning that the user may input information into them, but some text fields are *read-only.* A text field which accepts only integer values is called a *numeric* field. Read-write and numeric text fields can obtain the focus of the cursor by using the `Tab` key. Pressing `Return` while the focus is on a read-write text field will generally cause an update procedure to be executed. Usually, this saves the appropriate variables for that window to the Postmaster. In some cases, other window items will be updated to reflect new information.
- Message: A message is similar to a text field. The program uses messages to display information to the user. A message is always read-only.
- Settings: A setting is a list of choices. A *non-exclusive* setting presents options which may be chosen independently, such as selecting data to be saved into a file. An *exclusive* setting is a set of options which are mutually exclusive, such as the choices "on" and "off." Both exclusive and non-exclusive settings function the same way with respect to keyboard input. Pressing `Return` or `Space` while the focus is on a setting of these types will select the appropriate choice. A third type of setting is called a *stack* setting. It is related to the exclusive setting in that only one option may be selected from among the choices, but unlike the exclusive setting, not all options are displayed at one time. The currently selected option is displayed on the popup menu button; all options may be viewed by clicking the button with SELECT. Stack settings do not respond to keyboard input.
- Listbox: A listbox is a box with a list of entries in it, one or more of which may be selected by the user. Listboxes differ from exclusive and non-exclusive settings by the fact that the choices may not be completely independent or exclusive. For example, a listbox may allow only two of its entries to be selected. Also, the number of entries that can be selected in a listbox might change with the value of another variable. Selected entries will appear differently, usually with a raised gray background. Pressing SELECT or `Space` will (de)select the entry that the mouse is pointing to or the underlined entry, respectively. The underlined entry may be moved up or down with the arrow keys. Pressing `Return` will do nothing.
- Text Pane: A text pane is a region in which the user may read, write, and edit text. Some text panes are *read-only.*

## 2.2 The Command Window

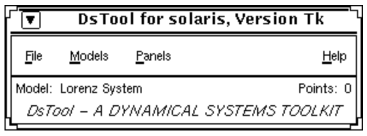

DsTool for solaris, Version Tk
File
Models
Panels
Help
Model: Lorenz System
Points: 0
DsTool – A DYNAMICAL SYSTEMS TOOLKIT

Figure 2.2: The Command window.

**Window title:** DsTool for (architecture), Version Tk

**Function:** The Command window is the main control panel for the dynamical system toolkit.

**Description:** The Command window is opened at start-up. The main function of this window is to allow the user to interact with the package through various panel items.

**Panel items:**

- File menu button:

  **Save:** Opens and brings to the foreground the Save window.

  **Load:** Opens and brings to the foreground the Load window.

  **Print:** Opens and brings to the foreground the Print window.

  **Exit:** Exits DsTool.

- Models menu button:

  **Mappings:** Allows the user to choose a mapping as the dynamical system to study.

  **Vector Fields:** Allows the user to choose a vector field as the dynamical system to study.

  **Animated systems:** Allows the user to choose a vector field enabled with accompanying Geomview animation as the dynamical system to study.

  **User-defined categories:** Allows the user to choose a model from categories the user has defined in a custom version of DsTool as the dynamical system to study. Multiple categories can be defined by the user. See section 4.2.6 on installing dynamical systems in DsTool for information.

  **Parser:** Opens and brings to the foreground the Parser window.

- Panels menu button:

  **One-D:** Opens and brings to the foreground a 1-D View window.

  **Two-D:** Opens and brings to the foreground a 2-D View window.

  **Three-D(Geomview):** Opens and brings to the foreground the 3-D View window. The three dimensional viewing functions of this window require Geomview.

  **Animation(Geomview):** Opens and brings to the foreground the animation window. The animation features of this window require Geomview and that the model be either from the "Animated systems" category or user-defined and enabled for animation. See the accompanying documentation in `$DSTOOL/my_dstool/README` for information on enabling models for animation.

  **Selected:** Opens and brings to the foreground the Selected Point window.

  **Function:** Opens and brings to the foreground the Function Values window.

  **Defaults:** Opens and brings to the foreground the Defaults window.

  **Browser:** Opens and brings to the foreground the Browser window.

  **Orbits:** Opens and brings to the foreground the Orbits window.

**Multiple:** Opens and brings to the foreground the Multiple Orbits window.

**Fixed Points:** Opens and brings to the foreground the Fixed Points window.

**Periodic Orbits:** Opens and brings to the foreground the Periodic Orbits window.

**Equilibrium Continuation:** Opens and brings to the foreground the Equilibrium Continuation window.

**User-Defined:** Opens and brings to the foreground any user-installed windows. See the accompanying documentation in `$DSTOOL/my_dstool/README` for information on how to add custom analysis code to DsTool.

- Help menu button:

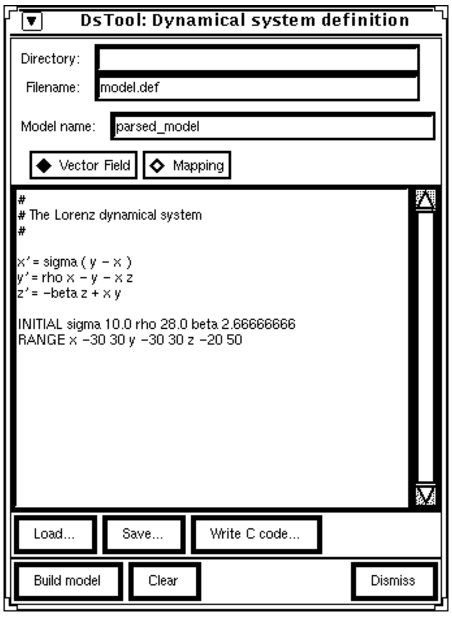
  **Model:** Opens and brings to the foreground the Model Help window.

  **File:** Opens and brings to the foreground the Files window.

  **About DsTool:** Opens and brings to the foreground the About DsTool Help window.

- Model name message: Displays the name of the current dynamical system.
- Points message: Displays the total number of data points (of all types) currently stored in memory.
- status message: Displays messages about computations in progress or other helpful messages. By default, the program name is displayed.

## 2.3 The Parser Window



Filename: model.def

Model name: parsed_model

```
#
# The Lorenz dynamical system
#

x' = sigma ( y - x )
y' = rho x - y - x z
z' = -beta z + x y

INITIAL sigma 10.0 rho 28.0 beta 2.66666666
RANGE x -30 30 y -30 30 z -20 50
```

Figure 2.3: The Parser window.

**Window title:** DsTool: Dynamical system definition

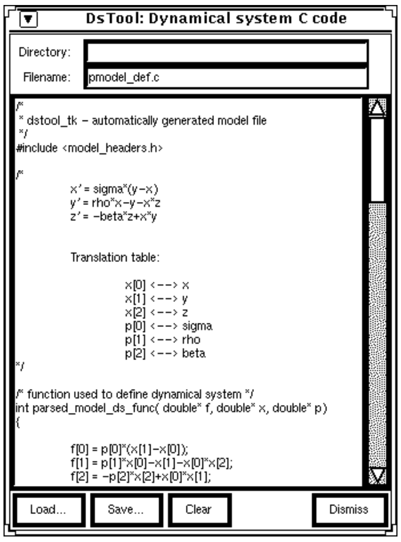
**Function:** The Parser window allows the user to define a dynamical system without writing C code and compiling a new model into DsTool.

**Description:** The Parser window is opened by selecting the Parser option from the Models menu button located in the Command window. The user can load or save model (vector field or mapping) definitions, build the current model into DsTool as the current dynamical system, or generate C code which can be subsequently compiled into DsTool.

**Panel items:**

- Directory read-write text field: Displays the currently selected directory which DsTool will use to save to or read from. The default is the current value of the UNIX environmental variable `DSTOOL_DATA_DIR`, or if that is not set, the current working directory.
- Filename read-write text field: Displays the currently selected filename which to write to or read from. This should be a single word of no more than 30 characters. The default value is "model.def".
- Model name read-write text field: Displays the currently selected string describing the model. The default value is "parsed_model".
- Vector Field / Mapping exclusive selection: Allows the user to choose whether the dynamical system should be interpreted as a vector field or a mapping. The default is to interpret it as a vector field.
- Dynamical system definition text pane: Displays the definition for the dynamical system in simplified notation. See section 3.1 for more information on how to define a dynamical system in this section. The default is to display text describing the Lorenz system.
- Load window button: Opens and brings to the foreground the File window for loading the model file specified by the Filename text field, in the directory specified by the Directory text field.
- Save window button: Opens and brings to the foreground the File window for saving the model definition given in the dynamical system definition text. The model definition will be saved to the directory specified by the Directory text field under the filename specified by the Filename text field.
- Write C code command-window button: Parses the textual information in the dynamical system definition text pane and generates C code, then opens and brings to the foreground the C Code window.
- Build model command button: Incorporates the model defined in the dynamical system definition text pane into DsTool as the current model. If the model definition is inadequate, an error message is printed.
- Clear command button: Clears the dynamical system definition text pane.
- Dismiss command button: Closes the Parser window.

## 2.4 The C Code Window



```
/*
 * dstool_tk - automatically generated model file
 */
#include <model_headers.h>

/*
        x' = sigma*(y-x)
        y' = rho*x-y-x*z
        z' = -beta*z+x*y


        Translation table:

                x[0] <--> x
                x[1] <--> y
                x[2] <--> z
                p[0] <--> sigma
                p[1] <--> rho
                p[2] <--> beta
*/

/* function used to define dynamical system */
int parsed_model_ds_func( double* f, double* x, double* p)
{

        f[0] = p[0]*(x[1]-x[0]);
        f[1] = p[1]*x[0]-x[1]-x[0]*x[2];
        f[2] = -p[2]*x[2]+x[0]*x[1];
```

Figure 2.4: The C Code window.

**Window title:** DsTool: Dynamical system C code

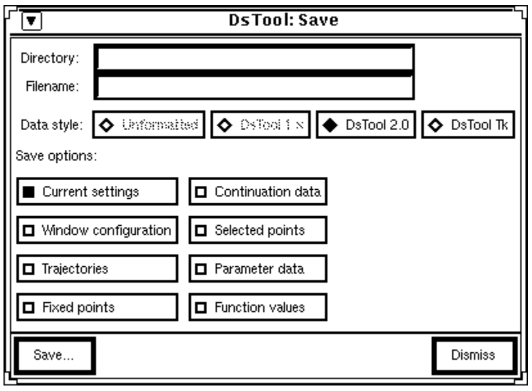
**Function:** The C Code window allows the user to edit, save or load automatically generated C code.

**Description:** The C Code window is opened by selecting the Write C Code command/window button located in the Parser window. The user can inspect the C code which has been generated by the Parser, edit it, and save it to a file. In addition, C code can be loaded from previously generated model files.

**Panel items:**

- Directory read-write text field: Displays the currently selected directory which DsTool will use to save to or read from. The default is the string in the Directory text field of the Parser window.
- Filename read-write text field: Displays the currently selected filename which to write to or read from. This should be a single word of no more than 30 characters followed by the standard ".c" suffix. The default string is "pmodel_def.c".
- C code definition text pane: Displays the C code for the dynamical system. See section 3.1.6 for information on how the C code was generated.
- Load window button: Opens and brings to the foreground the Files window for loading C code datafiles.
- Save window button: Opens and brings to the foreground the Files window for saving the C code datafile.
- Clear command button: Clears the C code definition text pane.
- Dismiss command button: Closes the C Code window.

## 2.5 The Save Window


DsTool: Save

Directory:
Filename:

Data style: ◇ Unformatted ◇ DsTool 1.x [✓ ◆ DsTool 2.0] ◇ DsTool Tk

Save options:

[✓ ■ Current settings] □ Continuation data
□ Window configuration □ Selected points
□ Trajectories □ Parameter data
□ Fixed points □ Function values

Save... Dismiss

Figure 2.5: The Save window.

**Window title:** DsTool: Save

**Function:** The Save window is used to control saving memory data to files.

**Description:** The Save window is opened by selecting the Save option from the File menu button located in the Command window. The user can specify parts of DsTool memory to save.

**Panel items:**

- Directory read-write text field: Displays the currently selected directory to which DsTool will save data. The default is the current value of the UNIX environmental variable `DSTOOL_DATA_DIR`, or if that is not set, the current working directory.
- Filename read-write text field: Displays the currently selected filename to which DsTool will save data. The default is an emptry string.
- Data style exclusive selection:

  **DsTool Tk Format:** Specifies that the data should be saved in DsTool Tk format. This format is currently not completely operational. The only options which can be saved with this format are "Current Settings" and "Window Configuration".

  **DsTool 2.0 Format:** Specifies that the data should be saved in DsTool 2.0 format. This format allows all of the options below to be saved. This is the default setting.
- Save options non-exclusive selections:

  **Current Settings:** Specifies that the current initial conditions, parameter values, and default settings are saved to the file specified by the Filename text field. Specifically, the data saved is a complete description of the Postmaster, with the exception of the "Models" object. No Tcl variables which control the look of the user interface (such as COLOR and EXT) are saved. This is selected by default.

  **Window Configuration:** Specifies that the positions and sizes of all DsTool windows are saved to the data file specified by the Filename text field. For each 2-D View window that is open, the view options are also saved. View options include the horizontal and vertical coordinates and plotting ranges. By saving the current settings and configuration before ending a DsTool session, the user may preserve virtually all the program working environment. Later, the user can restart DsTool, load the saved configuration and data file and continue working essentially as if the session had not been terminated.

**Trajectories:** Specifies that all trajectories (whether or not they are currently displayed) are saved to the data file specified by the Filename text field. Be warned that by using this option it is easy to generate massive data files. Flow objects (generated using the Multiple Orbits window) are also saved when this option is chosen.

**Fixed points:** Specifies that all fixed points (which may be found using the Fixed Points window) are saved to the data file specified by the Filename text field. Any computed stable or unstable manifolds are also saved.

**Continuation data:** Specifies that all continuation trajectories (whether or not they are currently displayed) are saved to the data file specified by the Filename text field.

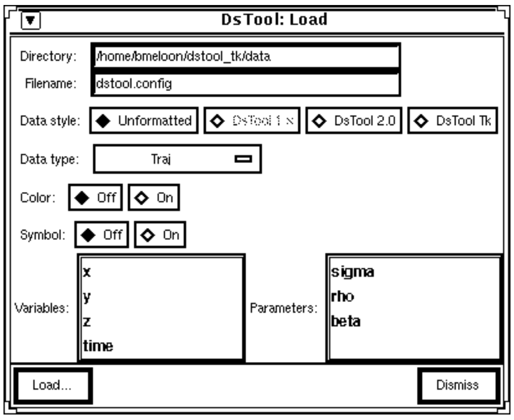
**Selected points:** Specifies that selected point data be saved to the data file specified by the Filename text field. Selected Point data contains elements of the cross-product of phase and parameter space.

**Parameter data:** Specifies that parameter data be saved to the data file specified by the Filename text field. Parameter data and selected point data are similar in that they both contain elements of the cross-product of phase and parameter space. The main difference lies in how these data sets are displayed in view windows. Parameter data is only displayed in a view window if all coordinates are parameters. In fact, parameter data does not contain any phase space variable information. This version of DsTool does not save information into the Parameter memory, but this option is included for compatibility with other versions.

**Function values:** Specifies that function values be saved to the data file specified by the Filename text field. Function values are not stored in memory but they can be written to a data file. Function values are *never* read back into DsTool; DsTool will stop reading when it encounters a function value.

- Save window button: Opens and brings to the foreground the Files window, for saving the requested data to the file.
- Dismiss command button: Closes the Save window.

## 2.6 The Load Window


DsTool: Load

Directory: /home/bmeloon/dstool_tk/data
Filename: dstool.config
Data style: [✓ Unformatted] DsTool 1.x DsTool 2.0 DsTool Tk
Data type: Traj
Color: [✓ Off] On
Symbol: [✓ Off] On
Variables: x y z time
Parameters: sigma rho beta

Figure 2.6: The Load window.

**Window title:** DsTool: Load

**Function:** The Load window allows the user to load data from an external file into memory.

**Description:** The Load window is opened by selecting the Load option from the File menu button located in the Command window. The user can select a file and load it into DsTool's memory.

**Panel items:**

- Directory read-write text field: Displays the currently selected directory from which DsTool will load data. The default is the value of the UNIX environmental variable `DSTOOL_DATA_DIR`, or if that is not set, the current working directory.
- Filename read-write text field: Displays the currently selected file from which DsTool will load data. The default is the file "dstool.config".
- Data style exclusive setting:

  **DsTool Tk Format:** Specifies that the data file was saved by DsTool Tk in DsTool Tk format. If the load window was elongated (see below), then selecting this option will return the load window to its abbreviated size.

  **DsTool 2.0 Format:** Specifies that the data file was saved by DsTool 2.0 or by DsTool Tk in DsTool 2.0 format, or is in a format which DsTool can read. (The DsTool format is discussed at greater length in the DsTool Reference Manual.) This is the default load option. If the load window is elongated (see below), then selecting this option will return the load window to its abbreviated size.

  **Unformatted:** Specifies that the data file was created by an application which wrote an ASCII data file. The data can be read into any memory object. When this option is chosen, the load window lengthens to reveal additional panel items which control the loading of unformatted data.
- Load window button: Opens and brings to the foreground the Files window, for loading the requested file into DsTool. In the event that the data does not match the data structure of the current installed system (*e.g.*, dimensions wrong), an error message is displayed. If the Unformatted option is selected, then the data is loaded according to the specified unformatted options (described below).

  There are seven types of formatted data which DsTool can load: current settings data, configuration data, trajectory data, fixed point data, continuation data, parameter data, and selected point data. Current setting data is a mirror of the non-memory and non-function items of the Postmaster. Configuration data consists of the size and placement of DsTool windows, as well as viewing options for 2-D View windows. The remaining data types are discussed briefly in the Save window section of this document (and in detail in the Reference Manual). We note here that auxiliary function values are special in that they are not stored in memory but are derived from parameter and variable values whenever they are needed.

  The following panel items are visible only when the user chooses the Unformatted option setting.
- Data type stack setting: Allows the user to choose which data type will be filled with the unformatted data. Unformatted data may be loaded into one of six data types. The possible data types are trajectory (Traj), multiple trajectory (Mult), fixed point (Fixed), continuation (Continuation), parameter (Param), and selected point (Sel_Pt) data. The default setting is trajectory.
- Color exclusive setting: Allows the user to choose whether or not data should be considered to have color information. Each data object in DsTool has two colors associated with it: an "alternating color" and a "pick color". See section 3.3 for more information on color modes. All unformatted data is assigned the current alternating color. If the data does *not* contain color-coding information, then the data is assigned the current pick color; otherwise, the pick color is determined by the color-coding information. The default is to not have color information.

Within an unformatted data file, color is coded as an integer. The integer appears immediately after all variable and parameter values. If this integer is $k$, then the associated data object will be color-coded with the $k$th color of the currently installed colormap. Obviously, $0 \le k \le N$ where $N$ is the maximum number of colors in the current colormap. As an example, if the user has specified that an unformatted data file contains fixed point data, values for the variables $x$ and $y$, and color-coding information, then if the file has the line

```
0.0  0.5  12
```

then DsTool will store the data point $(x, y) = (0.0, 0.5)$ and whenever a view window has the pick color option selected, the point $(0.0, 0.5)$ will be plotted in the 12th color of the colormap.

Trajectories (and multiple trajectories) cannot currently be assigned colors from within an unformatted data file; they automatically receive the current pick color. All other data objects can be assigned colors.

- Symbol exclusive setting: Allows the user to choose whether or not data should be considered to have symbol-coding information. If the data does *not* contain symbol-coding information, then the data is assigned the current symbol (visible in the Defaults window); otherwise, the symbol is determined by the symbol-coding information. The default is to not have symbol information.

  Within an unformatted data file, a symbol is coded as an integer. The integer must be the last item in the field which defines the data object. If this integer is $k$, then the associated data object will be symbol-coded with the $k$th symbol. Currently, valid symbols require $0 \le k \le 11$. The values $0 \le k \le 2$ specify dots; $3 \le k \le 5$ specify crosses; $6 \le k \le 8$ specify boxes; and $9 \le k \le 11$ specify triangles. In each case, there are three possible sizes, with the smaller values corresponding to smaller symbols.

  As an example, if the user has specified that an unformatted data file contains fixed point data, values for the variables $x$ and $y$, and symbol-coding information, and if the file has the line

  ```
  0.0  0.5  8
  ```

  then DsTool will store the data point $(x, y) = (0.0, 0.5)$ and the point will be plotted with the 8th symbol, which happens to be the largest box. If the user wanted both color *and* symbol information (say the 12th color and the 8th symbol), then the data file might read something like this:

  ```
  0.0  0.5  12  8
  ```

  As with colors, trajectories cannot currently be assigned symbols.

- Variables listbox: Allows the user to choose which variables will be loaded with data from the data file. The listbox contains all of the variables of the current dynamical system, including the independent variable. Any or all of these variables may be selected independently of each other. Unformatted data is loaded into variables in the same order as the variables appear on the list: the first number encountered is stored in the first selected variable on the list, the second number is stored on the second variable on the list, and so on. If $k$ of these variables are selected and if $j$ parameters are selected, then DsTool will load the first $k$ pieces of data into the selected variable, load the next $j$ pieces of data into the selected parameters, and then repeat the process until it reaches the end of the data. Any variables not selected are assigned their current value (visible in the Selected Point window). By default, none of the variables are selected.

- Parameters listbox: Allows the user to choose which parameters will be loaded with data from the data file. The listbox contains all of the parameters of the current dynamical system. Any or all of these parameters may be selected independently of each other. When the file is loaded, DsTool proceeds as above. Unformatted data is loaded into parameters in the same order as the parameters appear on the list. Any parameters not selected are assigned their current value (visible in the Selected Point window). By default, none of the parameters are selected.

As an example, suppose a dynamical system has variables $(x, y, z)$ and parameters $(\alpha, \beta, \gamma)$. If the current values of these quantities are $(x_0, y_0, z_0)$ and $(\alpha_0, \beta_0, \gamma_0)$ and if the user specifies that an unformatted data file contains values for $x$, $z$, and $\gamma$, then a line in the file of the form

```
8.3  -2.3 0.01
```

will create a data object of the requested type and this data object will contain the variable data $(8.3, y_0, -2.3)$ and the parameter data $(\alpha_0, \beta_0, 0.01)$. Associated to the data object will be the current pick point color and the current symbol, since these quantities were not specified.

- Dismiss command button: Closes the Load window.

## 2.7 The Files window

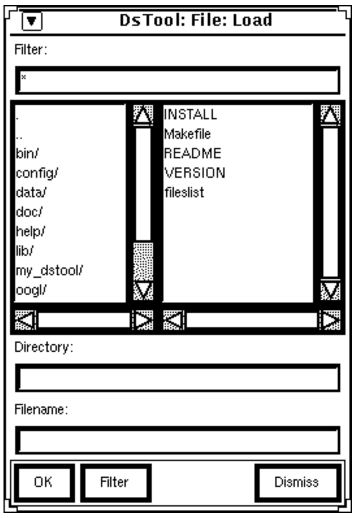


Figure 2.7: The Files window.

**Window title:** DsTool: File: (Action)

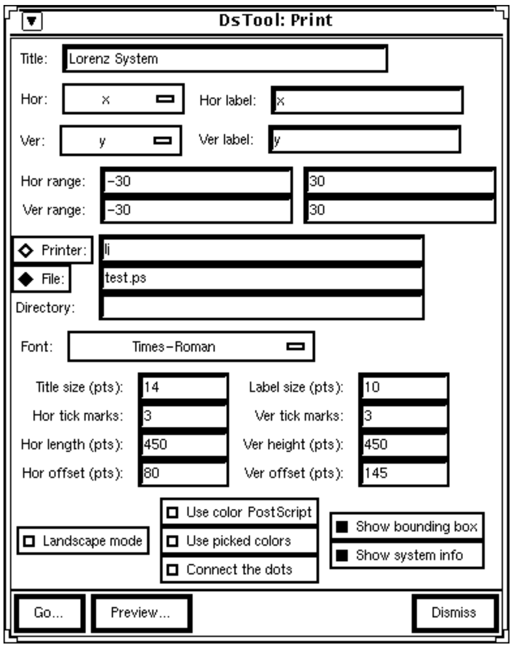
**Function:** The File window is used to control the exchange of data between DsTool and outside files.

**Description:** The File window is opened by selecting any button which requires interaction with outside files. Specifically, these buttons are the Load and Save window buttons located in the C Code and Parser windows, the File option from the Help menu button located in the Command window, the Load window button in the Load window, the Save window button in the Save window, the Go window button in the Print window when the exclusive setting is "File", and the Go window button in the Snapshot window. In each case, the Files window provides the user with a convenient way to save or load a file.

**Panel items:**

- Filter read-write text field: Displays a string which is used to control the display of files in the Files listbox. Only those files which match the string will be displayed. The string should use the usual UNIX wildcards: asterisk, question mark, etc. If the field is blank, no files will be displayed. If the field contains only an asterisk, then all of the files in the specified directory will be displayed. Pressing `Return` or pressing the Filter command button will apply the filter to the Files listbox. The default value for this text field depends upon which button caused the creation of the Files window.
- Directory listbox: Allows the user to choose a directory from which to select a file. The listbox contains the current directory, the parent directory, and a list of the subdirectories of the current directory. Double clicking on a directory name in the listbox changes the current directory to the selected one. The entries in the Files listbox change accordingly.
- Files listbox: Allows the user to choose a file in the given directory. The listbox contains a list of the files contained in the current directory which match the filter string. Double clicking on a file causes it to be selected for the action of the window. If the action is saving to a file, DsTool will ask for confirmation before overwriting the file. If the action is loading a file, DsTool will load the file.
- Directory read-write text field: Displays the current working directory. Pressing `Return` while the cursor is in this field updates the Directory and Files listboxes. The default value of this text field depends upon which button caused the creation of the Files window.
- Filename read-write text field: Displays the current filename, which will be used for the specified action if the OK button is pressed. The default value of this field depends upon which button caused the creation of the Files window.
- OK command button: Causes the specified action (save or load) to be performed, using the current filename and directory. If the action is "Save" and the file doesn't exist, DsTool will create and write to the file. If the file does exist, DsTool will ask for confirmation before overwriting the file. If the action is "Load" and the file doesn't exist, an error message will be printed. If the file does exist, DsTool will load the file.
- Filter command button: Causes the specified filter to be applied to the files which appear in the Files listbox.
- Dismiss command button: Closes the Files window.

## 2.8 The Print Window


DsTool: Print

Title: Lorenz System

Hor: x  Hor label: x

Ver: y  Ver label: y

Hor range: -30  30

Ver range: -30  30

◇ Printer: li

◆ File: test.ps

Directory:

Font: Times-Roman

Title size (pts): 14  Label size (pts): 10

Hor tick marks: 3  Ver tick marks: 3

Hor length (pts): 450  Ver height (pts): 450

Hor offset (pts): 80  Ver offset (pts): 145

☐ Landscape mode

☐ Use color PostScript

☐ Use picked colors

☐ Connect the dots

■ Show bounding box

■ Show system info

Figure 2.8: The Print window.

**Window title:** DsTool: Print

**Function:** The Print window allows the user to output data to a file or a printing device which understands PostScript.

**Description:** The Print window is opened by selecting the Print option from the File menu button located in the Command window, or by selecting the Print option from the Options menu button located in any 2-D or 1-D View window. The user may select printing options, and print the output to a printer or a file. In addition, after the file has been written to a file, DsTool can call an external program to view the output.

**Panel items:**

- Title read-write text field: Displays the string which will appear at the top of the printed image. The default is the title of the dynamical system as it appears on the Model name message of the Command window.
- Hor variable stack setting: Allows the user to choose the abscissa variable for the plot. The default value for this field depends upon where the Print window was opened from. If the Print window is called from the Command window, the first variable is used. If the Print window is called from a 2-D or 1-D View window, the current value of the horizontal variable is used.
- Hor label read-write text field: Displays the string which will appear centered below the horizontal axis of the printed image. The default is the string currently being used to indicate the abscissa. The value will change when the Hor variable stack setting is changed.
- Ver variable stack setting: Allows the user to choose the ordinate variable for the plot. The default value behaves the same way as for the horizontal variable, with the second variable being used as the default value when the Print window is opened from the Command window.

- Ver label read-write text field: Displays the string which will appear centered to the left of the vertical axis of the printed image. The default is the string currently being used to indicate the ordinate. The string will change when the Ver variable stack setting is changed.
- Hor range read-write fields: Display the min and max values for the abscissa. The default value for this field depends upon where the Print window was called from. If the Print window is called from the Command window, the default values for the horizontal variable's range are used. If the Print window is called from a 2-D or 1-D View window, the current values of the horizontal variable's range are used.
- Ver range read-write fields: Display the min and max values for the ordinate. The default value behaves the same way as for the horizontal variable's range.
- Printer / File exclusive setting: Allows the user to choose whether the output is written to a file or piped to a printer. The default is to print to a file. The read-write text field immediately to the right of the Printer option specifies the printer to which the plot will be sent. The default printer name is specified by the environmental variable LPDEST if set, by the environmental variable PRINTER if LPDEST is not set, or the default printer name, "lp". The read-write text field immediately to the right of the File radiobutton specifies the filename of the PostScript file to which data is saved. The default is the file "test.ps".

  A common experience is to attempt to print a PostScript file which is too big to fit entirely within a printer's buffer. In this case, we recommend attempting to print the file with the UNIX command `lpr -s` *filename*. We remark in any case that piping data directly to a printer is somewhat riskier than creating a PostScript file and then sending that file to the printer, chiefly because if the pipe fails for any reason, then the image may be lost.
- Directory read-write text field: Displays the directory to which the file will be saved. This field works in conjunction with the filename field directly above it. The default value is the current working directory.
- Font stack setting: Allows the user to choose the font to print the axes labels and title in. The allowed fonts are detailed (with examples for each font typeface) in Appendix A. The default value is "Times-Roman".
- Title size (Pts) numeric field: Displays the size (in printer's points; one point is approximately 1/72 inches) of the title. The range is from 4 points to 30 points. The default is 14 points.
- Label size (Pts) numeric field: Displays the size (in printer's points) of the axis labels. The range is from 4 points to 30 points. The default is 10 points.
- Hor tick marks numeric field: Displays the number of tick marks on the horizontal axis of the bounding box. If Show bounding box is off, this field is not used. The default value is 3.
- Ver tick marks numeric field: Displays the number of tick marks on the vertical axis of the bounding box. If Show bounding box is off, this field is not used. The default value is 3.
- Hor length (pts) numeric field: Displays the width in points of the picture. The default value is 450.
- Ver height (pts) numeric field: Displays the height in points of the picture. The default value is 450.
- Hor offset (pts) numeric field: Displays the horizontal indentation in points of the lower left corner of the picture. The default value is 80 when not in Landscape mode, and 575 when in Landscape mode.
- Ver offset (pts) numeric field: Displays the vertical height from the bottom of the page in points of the lower left corner of the picture. The default value is 145.

- Landscape mode exclusive setting: Allows the user to choose whether to print in landscape mode (rotated by 90 degrees) or not. The default value is no.
- Use color PostScript exclusive setting: Allows the user to choose between printing a black and white PostScript image (the default) and a color image. Warning: DsTool does not check to see if the selected printer is a color printer. Printing a color image on a non-color printer will lead to images of poor quality.
- Use picked colors exclusive setting: Allows the user to choose whether to print with picked colors or not. The default is to not use picked colors, even if the 2-D View window from which Print is called has pick colors chosen. If the Use color postscript setting is off, this field is not used.
- Connect the dots exclusive setting: Allows the user to choose whether consecutive points in continuous data should be drawn with connecting line segments. The default is to not connect the dots.
- Show bounding box exclusive setting: Allows the user to choose whether a bounding box will be printed about the plotted points. The bounding box will contain tick marks as specified by the Hor and Ver Tick Marks numeric fields (see above). The default is to show the bounding box.
- Show system info exclusive setting: Allows the user to choose whether the upper left portion of the printed image will contain information about the dynamical system being plotted. This information includes the name of the dynamical system, the time that the image (or PostScript file) was generated, the range of the ordinate and abscissa, the initial settings (as seen in the Selected Point window), and the number of points in the plot. If the dynamical system being plotted is a vector field, then the system information also includes the value for Step size as it appears on the Orbits window. The default value is to show the system information.
- Go command or window button: Sends the current data file to the printer to be printed, or opens and brings to the foreground the Files window for saving the data to a file. Its function depends upon the value of the Printer/File exclusive setting.
- Preview command/window button: Invokes an external program to view the output of the Print routines. This option requires that the output be saved to a file before trying to preview. The default external program is "gs". See the accompanying documentation in `my_dstool/README` for information on how to change this default program.
- Dismiss command button: Closes the Print window.

## 2.9 The 1-D View Window

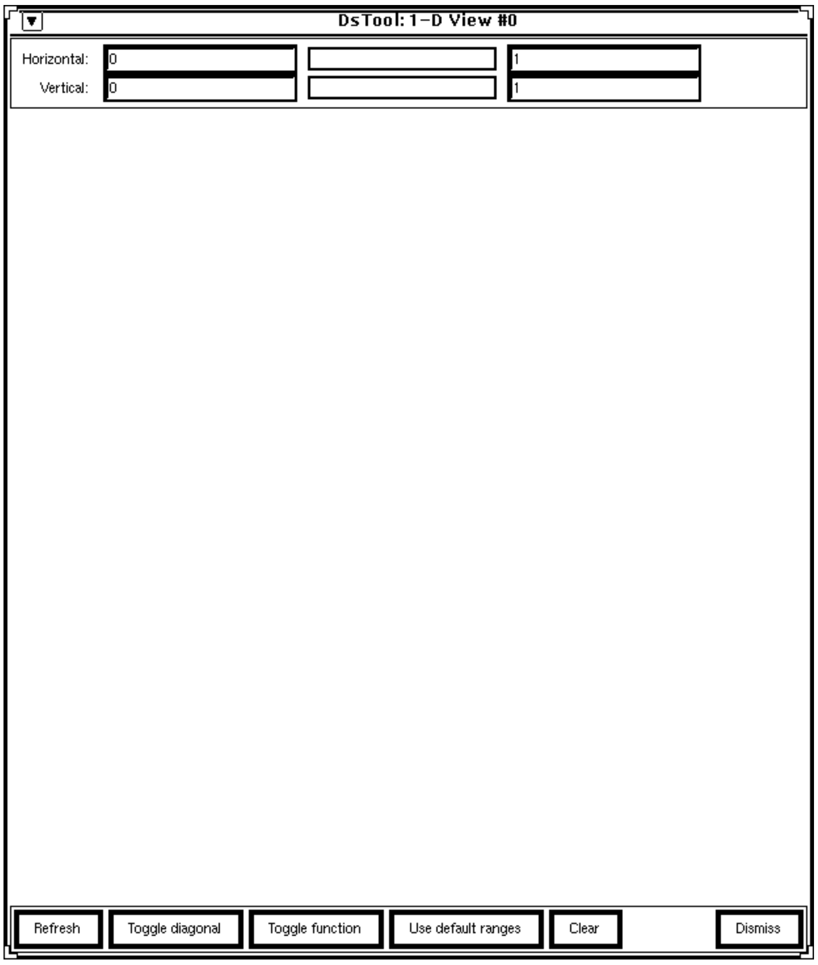


Figure 2.9: The 1-D View window.

**Window title:** DsTool: 1-D View $\#n$

**Function:** The 1-D View window graphically displays computed data.

**Description:** The 1-D View window is opened by selecting the 1-D View option from the Panels menu button located in the Command window. The 1-D View window can only be opened when the current dynamical system is a one-dimensional mapping, $f : \mathbb{R} \to \mathbb{R}$. Multiple 1-D View windows can be open simultaneously. All 1-D View windows display the same data, but they may use different scalings.

The window displays the "cobweb diagram" for trajectories of the system. That is, given an initial point $x_0$, we draw a vertical line from $(x_0, 0)$ to $(x_0, f(x_0))$, and then a horizontal line to $(f(x_0), f(x_0))$. We then repeat the process starting for $x_1 = f(x_0)$ drawing lines from $(x_1, x_1)$ to $(x_1, f(x_1))$ to $(f(x_1), f(x_1))$, and so forth.

When the window is open, data is displayed as it is computed. When the user resizes the window, the current variable ranges are maintained, but the scaling is adjusted accordingly.

As the user moves the mouse in the drawing canvas, the current coordinates of the mouse position are displayed in the window's panel. This region is blank when the mouse does not point into the canvas. Mouse events in the main drawing canvas have the following effects:

- The SELECT button sets the direction of the iterator to "forward." It also reads the values indicated by the cursor and adjusts the value of the variable in the Selected Point window.
- Dragging the SELECT button while holding down SHIFT creates an elastic rectangle used to rescale the window. When the user releases SELECT, a notice appears with the options to enlarge the view (the default, also known as "zoom in"), shrink the view (also known as "zoom out"), or cancel the selection. If the enlarge option is selected, the scaling is adjusted so that the selected region fills the canvas. If the shrink option is selected, the view is rescaled so that the whole view is put into the selected rectangle. The values in the horizontal and vertical min and max text fields are updated automatically and a screen refresh is performed.
- The MENU button sets the direction of the iterator to "backward." It also reads the values indicated by the cursor and adjusts the value of the variable in the Selected Point window.
- Clicking the ADJUST button is equivalent to selecting the Continue button on the Orbits window.

**Panel items:**

- Horizontal message: Identifies the three text fields to the right of this message as referring to the horizontal variable: the phase space variable. They are from left to right: the Horizontal min read-write text field, the Mouse abscissa position read-only text field, and the Horizontal max read-write text field.
- Horizontal min read-write text field: Displays the current minimum value of the displayed range of the abscissa (the variable).
- Mouse abscissa position read-only text field: Displays the current abscissa value of the mouse. If the mouse does not point into the canvas, this region is blank.
- Horizontal max read-write text field: Displays the current maximum value of the displayed range of the abscissa (the variable).
- Vertical message: Identifies the three text fields to the right of this message as referring to the horizontal variable: the function value. They are from left to right: the Vertical min read-write text field, the Mouse ordinate position read-only text field, and the Vertical max read-write text field.
- Vertical min read-write text field: Displays the current minimum value of the displayed range of the ordinate (the function value).
- Mouse ordinate position read-only text field: Displays the current ordinate value of the mouse. If the mouse does not point into the canvas, this region is blank.
- Vertical max read-write text field: Displays the current maximum value of the displayed range of the ordinate (the function value).
- Refresh command button: Clears the drawing canvas and redraws the currently stored data as selected by the user in the above settings.
- Toggle diagonal command button: Toggles the drawing of the diagonal ($y = x$) on the plotting canvas.
- Toggle function command button: Toggles the drawing of the graph of the function $y = f(x)$ on the plotting canvas.
- Use default ranges command button: Sets the ordinate and abscissa to their default ranges, and then refreshes the plotting canvas.
- Clear command button: Clears trajectory data from memory, and refreshes the canvas according to the above settings.
- Dismiss command button: Closes the 1-D View window.

## 2.10 The 2-D View Window

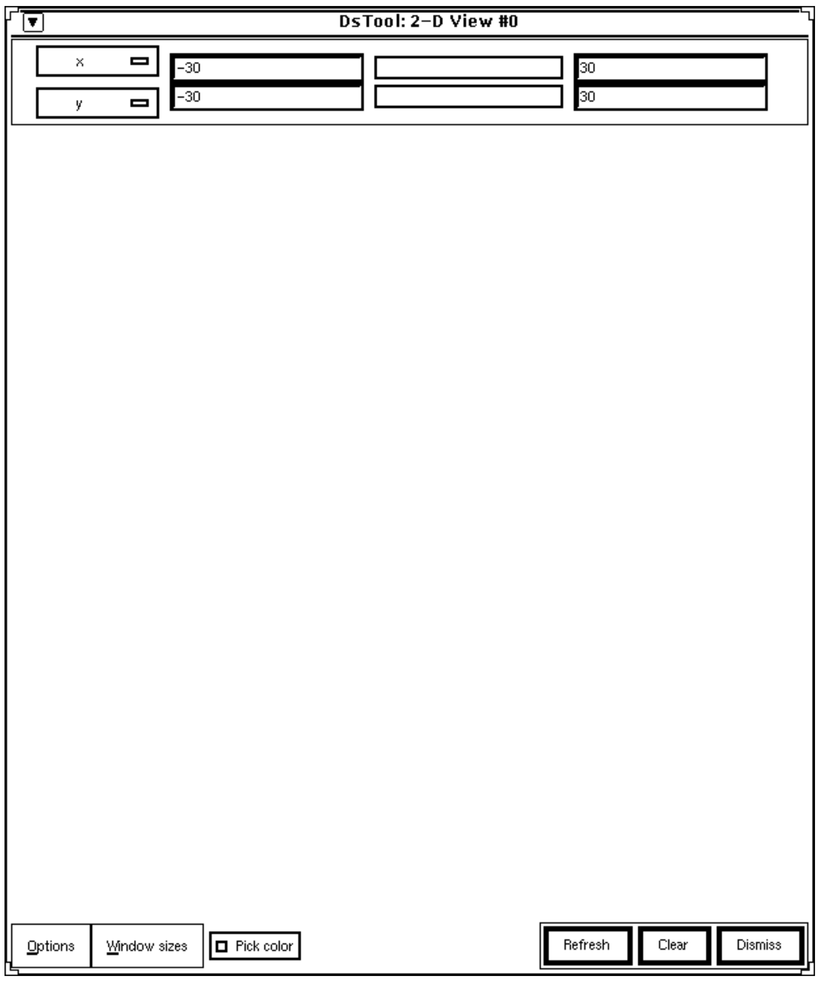


Figure 2.10: The 2-D View window.

**Window title:** DsTool: 2-D View #$n$

**Function:** The 2-D View window graphically displays computed data.

**Description:** The 2-D View window is opened by selecting the 2-D View option from the Panels menu button located in the Command window. Unlike the 1-D View window, a 2-D View window can always be opened, regardless of the number of phase space variables of the system. The user can select the plotting coordinates from all available variables, parameters, and auxiliary functions for the current dynamical system. Additionally, the range of each axis is displayed, and the size of the window and various other plotting options may be specified. Multiple 2-D View windows can be open simultaneously, and each may plot with different coordinates.

When the window is open, data is displayed as it is computed. When the user resizes the window, the current variable ranges are maintained, but the scaling is adjusted accordingly.

As the user moves the mouse in the drawing canvas, the current coordinates of the mouse position are displayed in the window's panel. This region is blank when the mouse does not point into the canvas. Mouse events in the main drawing canvas have the following effects:

- The SELECT button sets the direction of the integrator or iterator to "forward." It also reads the values indicated by the cursor. If one or both of these values are variables or parameters, these values are adjusted in the Selected Point window.
- Dragging the SELECT button while holding down SHIFT creates an elastic rectangle used to rescale the window. When the user releases SELECT, a notice appears with the options to enlarge the view (the default, also known as "zoom in"), shrink the view (also known as "zoom out"), or cancel the selection. If the enlarge option is selected, the scaling is adjusted so that the selected region fills the canvas. If the shrink option is selected, the view is rescaled so that the whole view is put into the selected rectangle. The values in the ordinate and abscissa Min and Max text fields are updated automatically and a screen refresh is performed.
- Dragging the SELECT button while holding down CONTROL creates an elastic rectangle used to input a region for the Multiple Orbits window. See Section 2.22 for details on this operation.
- The MENU button sets the direction of the integrator or iterator to "backward." It also reads the values indicated by the cursor. If one or both of these values are variables or parameters, these values are adjusted in the Selected Point window.
- Clicking the ADJUST button is equivalent to selecting the Continue button on the Orbits window.

**Panel items:**

- Hor stack setting: Allows the user to choose the coordinate of the abscissa. The user may choose between all variables (including the independent variable), all parameters, and all auxiliary functions. The three text fields to the right of the hor stack setting are the Hor min read-write text field, the Mouse abscissa position read-only text field, and the Hor max read-write text field.
- Hor min read-write text field: Displays the current minimum value of the displayed range of the abscissa.
- Mouse abscissa position read-only text field: Displays the current abscissa value of the mouse. If the mouse does not point into the canvas, this region is blank.
- Hor max read-write text field: Displays the current maximum value of the displayed range of the abscissa.
- Ver stack setting: Allows the user to choose the coordinate of the ordinate. The user has the same choices as for the Hor stack setting. The three text fields to the right of the ver stack setting are the Ver min read-write text field, the Mouse ordinate position read-only text field, and the Ver max read-write text field.
- Ver min read-write text field: Displays the current minimum value of the displayed range of the ordinate.
- Mouse ordinate position read-only text field: Displays the current ordinate value of the mouse. If the mouse does not point into the canvas, this region is blank.
- Ver max read-write text field: Displays the current maximum value of the displayed range of the ordinate.
- Options menu button:

  **Color:** Opens and brings to the foreground the Color window.

  **Continuation Colors:** Opens and brings to the foreground the Continuation Colors window.

  **Print:** Opens and brings to the foreground the Print window.

  **Snapshot:** Opens and brings to the foreground the Snapshot window.
- Window sizes menu button:

**Scale to fit:** Sets the abscissa and ordinate ranges so that all of the computed data points lie within a box centered in the canvas with a padding on each side. See section 2.18 for a description of how to control the parameter involved. The canvas is refreshed after the user selects this option.

**Set current to default:** Sets the abscissa and ordinate ranges to their default values. The canvas is refreshed after the users selects this option.

**Set default to current:** Sets the abscissa and ordinate default ranges to the current range.

- Pick color exclusive setting: Allows the user to choose whether points which are plotted in this 2-D View window will be plotted using their "pick color", or their "alternating color".
- Refresh command button: Clears the drawing canvas and redraws with the appropriate projection of currently stored data as selected by the user in the above settings.
- Clear command button: Clears trajectory data from memory, and refreshes the canvas according to the above settings.
- Dismiss command button: Closes the 2-D View window.

## 2.11 The Color Window

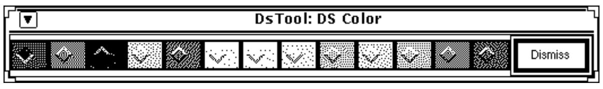


Figure 2.11: The Color window

**Window title:** DsTool: DS Color

**Function:** The Color window displays the colormap used to color trajectories.

**Description:** The Color window is opened by selecting the color option from the Options menu button located in any 2-D View window. The user can select the pick color for the next trajectory. The alternating color is automatically the next color in the colormap.

**Panel items:**

- *colors* exclusive setting: Allows the user to choose the pick color for the next computed trajectory. Each color in the colormap has its own setting in the array. See section 3.3 for more information on the use of colors in DsTool. The default selected color is the color farthest to the left, which in the standard colormap is "red".
- Dismiss command button: Closes the Color window.

## 2.12 The Continuation Colors Window

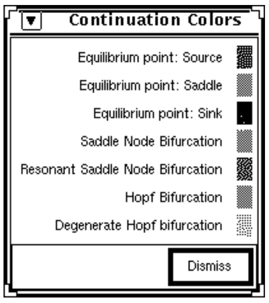

Continuation Colors
Equilibrium point: Source
Equilibrium point: Saddle
Equilibrium point: Sink
Saddle Node Bifurcation
Resonant Saddle Node Bifurcation
Hopf Bifurcation
Degenerate Hopf bifurcation
Dismiss

Figure 2.12: The Continuation Colors window

**Window title:** DsTool: Continuation Colors

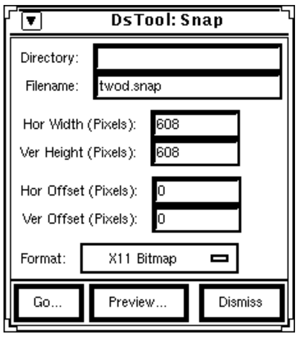
**Function:** The Continuation Colors window displays the colors used to color fixed points and bifurcation points.

**Description:** The Continuation Colors window is opened by selecting the Continuation Colors option from the Options menu button located in any 2-D View window.

**Panel items:**

- Bifurcation type messages: Display each type of fixed point and bifurcation point that DsTool computes, next to the color that DsTool uses to color those points. These colors follow the DsTool convention for hyperbolic fixed points: red = source, green = saddle, blue = sink. Saddle node bifurcation points are colored green. The remaining three bifurcation point types (resonant saddle, Hopf, and degenerate Hopf) are the last three entries in the colormap. In the default colormap, these colors are sea green, magenta, and orange, respectively. These colors can be changed by changing the colormap. See section 3.3 on colors in DsTool for information on changing the colormap.
- Dismiss command button: Closes the Continuation Colors window.

## 2.13 The Snapshot Window



Directory:
Filename: twod.snap
Hor Width (Pixels): 608
Ver Height (Pixels): 608
Hor Offset (Pixels): 0
Ver Offset (Pixels): 0
Format: X11 Bitmap

Figure 2.13: The Snapshot window.

**Window title:** DsTool: Snap

**Function:** The Snapshot window allows the user to store the image displayed in the canvas of a 2-D View window in an image file.

**Description:** The Snapshot window is opened by selecting the Snapshot option from the Options menu button located in any 2-D View window. The image that will be saved is the image in the 2-D View window from which the Snapshot window is opened. In order to save an image from a different 2-D View window, the Snapshot window must be closed, and re-opened from the other 2-D View window. The user may save the image to an image file in PPM, GIF, TIFF, or X11 Bitmap format, and view the output.

**Panel items:**

- Directory read-write text field: Displays the currently selected directory to which DsTool will save the image. The default is the current working directory.
- Filename read-write text field: Displays the currently selected filename to which DsTool will save the image. The default is the file "twod.snap".
- Hor Width (Pixels) numeric field: Displays the (horizontal) width of the image in pixels.
- Ver Height (Pixels) numeric field: Displays the (vertical) height of the image in pixels.
- Hor Offset (Pixels) numeric field: Displays the number of pixels to offset the image to the left. That is, if hor offset is set to a positive number, then the image in the 2-D View window will appear that number of pixels to the left in the saved image. If hor offset is set to a negative number, then the image in the 2-D View window will appear (the absolute value of) that number of pixels to the right in the saved image.
- Ver Offset (Pixels) numeric field: Displays the number of pixels to offset the image up. That is, if ver offset is set to a positive number, then the image in the 2-D View window will appear that number of pixels up in the saved image. If ver offset is set to a negative number, then the image in the 2-D View window will appear (the absolute value of) that number of pixels down in the saved image.
- Format stack setting: Allows the user to choose the file format in which to save the image. The allowed formats are: X11 Bitmap, PPM, GIF, and TIFF. The default format is X11 Bitmap. To save into GIF and TIFF formats, DsTool first saves into X11 Bitmap format, then calls an external program (the default is "convert") to convert to GIF or TIFF format. For information on changing this external program, see the accompanying documentation in `$DSTOOL/my_dstool/README`.
- Go window button: Opens and brings to the foreground the Files window to save the data to an image file.
- Preview command/window button: Invokes an external program which opens a window containing the image. The file must be saved before this option is used. The default program is "xv". For information on changing this external program, see the accompanying documentation in `$DSTOOL/my_dstool/README`.
- Dismiss command button: Closes the Snapshot window.

## 2.14 The 3-D View Window

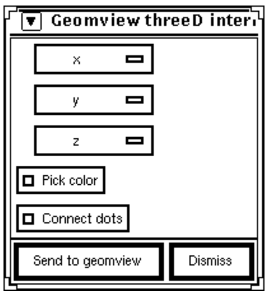


Figure 2.14: The 3-D View window.

**Window title:** DsTool: Geomview threeD interface

**Function:** The 3-D View window allows the user to graphically display computed data in Geomview.

**Description:** The 3-D View window is opened by selecting the Three-D (Geomview) option from the Panels menu button located in the Command window. The 3-D View window can always be opened, regardless of the number of phase space variables of the system, however only one 3-D View window may be opened. The user can select the plotting coordinates from all available variables, parameters, and auxiliary functions of the current dynamical system, or plotting coordinates can be set to "0".

Unlike the 2-D and 1-D View windows, data is not displayed as it is computed. The data must be computed first, and then sent to Geomview. In addition to the data points which are sent, Geomview will display a bounding box which corresponds to the product of the default ranges of the plotting coordinates.

**Panel items:**

- X-axis stack setting: Allows the user to choose the coordinate of the abscissa. The user may choose between all variables (including the independent variable), parameters, and auxiliary functions, or the setting "0". This last choice is useful for generating two or one dimensional plots in Geomview.
- Y-axis stack setting: Allows the user to choose the coordinate of the ordinate. The choices are the same as those for the X-axis.
- Z-axis stack setting: Allows the user to choose the coordinate of the vertical axis. The choices are the same as those for the X-axis.
- Pick color exclusive setting: Allows the user to choose whether points which are plotted in Geomview will be colored by the pick color or by the alternating color. The default is to use the alternating color.
- Connect dots exclusive setting: Allows the user to chooose whether points which are plotted in Geomview will be connected by lines or not. The default is to not connect the dots.
- Send to geomview command/window button: Invokes Geomview to display a three dimensional plot with the specified options.
- Dismiss command button: Closes the 3-D View window.

## 2.15 The Animation Window

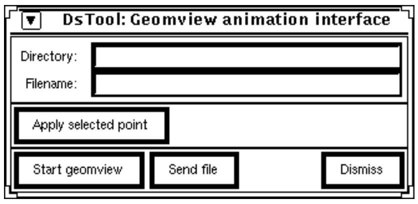


Figure 2.15: The Animation window.

**Window title:** DsTool: Geomview animation interface

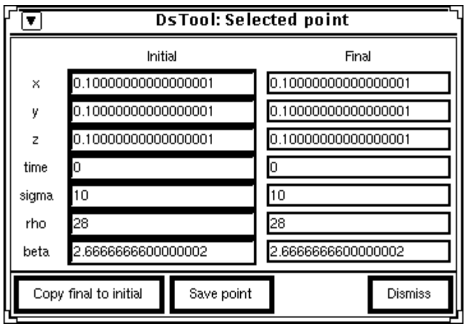
**Function:** The Animation window serves as an interface to Geomview, through which animations can be done.

**Description:** The Animation window is opened by selecting the Animation (Geomview) option from the Panels menu button located in the Command window. The window can always be opened, regardless of the number of phase space variables of the system. However, in order to use the animation features, the current dynamical system must be from the "Animated systems" submenu of the Models menu button located in the Command window, or must be enabled for animation by the user. In addition, Geomview must be available for use by the user. Unlike the 3-D View window, the animation will proceed in Geomview as the data is computed.

**Panel items:**

- Directory read-write text field: Displays the directory from which Geomview will take data to create the animation. The default directory is the current value of the UNIX environmental variable `DSTOOL_OOGL_DIR` if the current system is a standard animated system, or `MY_DSTOOL_OOGL_DIR` if the current system is a user-defined animated system. If the appropriate one of these variables is not set, the default is the subdirectory "oogl" of the UNIX environmental variable `DSTOOL`.
- Filename read-write text field: Displays the file from the above named directory from which Geomview will take data to create the animation. If the current dynamical system is an animated system, the default string is the value of the character string "*gv_filename" defined in the model file. If the current dynamical system is not an animated system, the default is an empty string.
- Apply selected point command button: Changes the position of the animated objects to correspond to the values in the selected point window.
- Start / Deactivate geomview command button: Starts Geomview if Geomview has not been previously started. Deactivates Geomview if Geomview has been previously started.
- Send file command button: Sends the file specified by the directory and filename fields to Geomview. If the animation objects do not currently appear in Geomview, they are created. If the animation objects do appear, they are reset to their original positions.
- Dismiss command button: Closes the Animation window.

## 2.16 The Selected Point Window


DsTool: Selected point

| | Initial | Final |
|---|---|---|
| x | 0.10000000000000001 | 0.10000000000000001 |
| y | 0.10000000000000001 | 0.10000000000000001 |
| z | 0.10000000000000001 | 0.10000000000000001 |
| time | 0 | 0 |
| sigma | 10 | 10 |
| rho | 28 | 28 |
| beta | 2.6666666600000002 | 2.6666666600000002 |

Figure 2.16: The Selected Point window.

**Window title:** DsTool: Selected Point

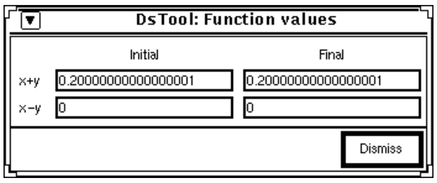
**Function:** The Selected Point window allows the user to examine and modify the current values of the variables and parameters.

**Description:** The Selected Point window is opened by selecting the Selected option from the Panels menu button located in the Command window. The two columns display the initial and final values of the variables and parameters for the last trajectory. (These are the same as the initial if there was no previous trajectory.) The values of auxiliary functions are displayed in the Function Values window (section 2.17).

**Panel items:**

- *Variablename* Initial read-write text field: Displays the initial value of the variable or parameter for the last trajectory. There is one read-write text field for each variable and parameter in the current dynamical system. The settings in this field will be the initial conditions for the next trajectory. These fields are updated upon clicking SELECT in a 2-D View window.
- *Variablename* Final read-only text field: Displays the final value of the variable or parameter for the last trajectory.
- Copy final to initial command button: Copies the information from the Final column to the Initial column.
- Save point command button: Creates a new point in memory, which contains the information from the Initial column. The point is created in DsTool's "Sel_Pt" Memory object.
- Dismiss command button: Closes the Selected Point window.

## 2.17 The Function Values Window


DsTool: Function values

| | Initial | Final |
|---|---|---|
| x+y | 0.20000000000000001 | 0.20000000000000001 |
| x−y | 0 | 0 |

Figure 2.17: The Function Values window.

**Window title:** DsTool: Function values

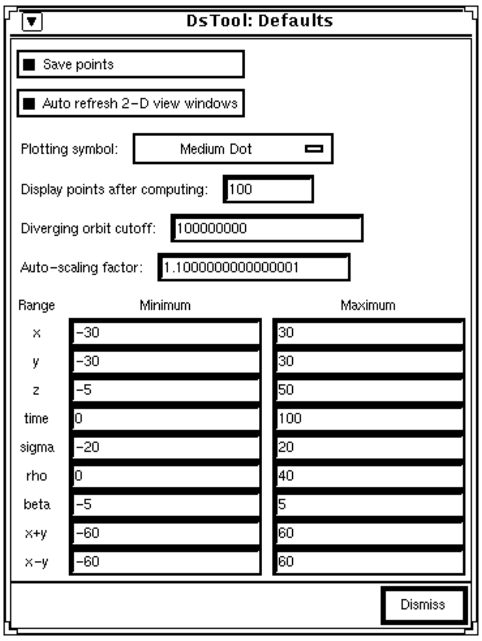
**Function:** The Function Values window displays the values of the auxiliary functions at the points specified by the Selected Point window.

**Description:** The Function Values window is opened by selecting the Function option from the Panels menu button located in the Command window. This window displays the values of the auxiliary functions at the final and initial points of the last trajectory. These are the points which are displayed in the Selected Point window described in section 2.16. If there are no auxiliary functions defined, a message to that effect is displayed.

**Panel items:**

- *Functionname* Initial read-only text field: Displays the value of the auxiliary function at the initial point of the last computed trajectory. There is one field for each auxiliary function defined for the current dynamical system.
- *Functionname* Final read-only text field: Displays the value of the auxiliary function at the final point of the last computed trajectory. There is one field for each auxiliary function defined for the current dynamical system.
- Dismiss command button: Closes the Function Values window.

## 2.18 The Defaults Window



■ Save points

■ Auto refresh 2-D view windows

Plotting symbol: Medium Dot

Display points after computing: 100

Diverging orbit cutoff: 100000000

Auto-scaling factor: 1.1000000000000001

| Range | Minimum | Maximum |
|---|---|---|
| x | -30 | 30 |
| y | -30 | 30 |
| z | -5 | 50 |
| time | 0 | 100 |
| sigma | -20 | 20 |
| rho | 0 | 40 |
| beta | -5 | 5 |
| x+y | -60 | 60 |
| x-y | -60 | 60 |

Figure 2.18: The Defaults window.

**Window title:** DsTool: Defaults

**Function:** The Defaults window allows the user to change some of DsTool's default settings.

**Description:** The Defaults window is opened by selecting the Defaults option from the Panels menu button located in the Command window. The user can change some of DsTool's values which control the computation and display of data. Changing a default value will not change data which has been previously computed, but will be used to compute future data, and to display all data.

**Panel items:**

- Save points exclusive setting: Allows the user to choose whether data is recorded or not. The default is to record data.
- Auto refresh 2-D View windows exclusive setting: Allows the user to choose whether 2-D View windows will be automatically updated when points are cleared from memory (via the "clear last" or "clear all" buttons in the Orbits window or the "delete" button in the Browser window), or the variables of the 2-D View window are changed. If it is set to off, the 2-D View windows will not be updated when the previous events occur. In either case, when new points are computed, the new points will be plotted on all existing 2-D View windows. The default is to auto refresh 2-D View windows.
- Plotting symbol stack setting: Allows the user to choose the symbol with which trajectories are plotted. It is possible to choose to plot with a dot, a cross, a triangle, and a box. Each of these types come in three sizes: small, medium, and large. The default symbol is "medium dot".
- Display points after computing numeric field: Displays the number of points to compute before storing and displaying points. Smaller values will cause slower plotting, due to the overhead of using X windows routines. Higher values will cause faster plotting, at the expense of not seeing the trajectory evolve in time. The default value is 100.
- Diverging orbit cutoff read-write text field: Displays the number DsTool will use to decide whether an orbit has diverged to infinity. If the distance from a point to the origin is greater than this number, the orbit containing that point is deemed to have diverged, and DsTool will stop calculation. The default value is $10^8$.
- Auto-scaling factor read-write text field: Displays the padding factor which determines how the auto-scaling feature on 2-D View windows sets the horizontal and vertical ranges. To find the auto-scaled range for a variable, DsTool first computes the minimum interval $R$ required to plot all previously computed data points for that variable. DsTool then resets the range to be the interval centered at the midpoint of $R$ with length given by the auto-scaling factor times the length of $R$. Consequently, an auto-scaling factor greater than 1 will allow all of the currently computed data to be plotted, whereas an auto-scaling factor less than 1 will exclude some of the data. The default value is 1.1.
- *Name* Minimum read-write text field: Displays the default minimum value for viewing the variable, parameter, or auxiliary function.
- *Name* Maximum read-write text field: Displays the default maximum value for viewing the variable, parameter, or auxiliary function.
- Dismiss command button: Closes the Defaults window.

## 2.19 The Browser Window

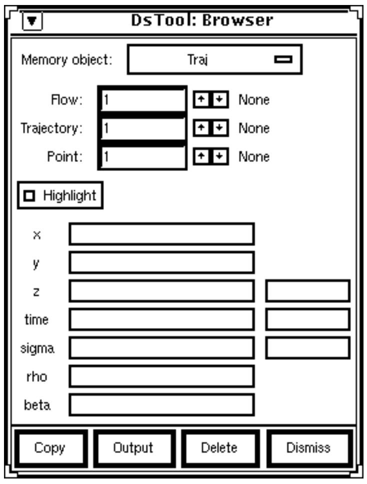


Figure 2.19: The Browser window.

**Window title:** DsTool: Browser

**Function:** The Browser window allows the user to examine data points stored in the DsTool memory objects.

**Description:** The Browser window is opened by selecting the Browser option from the Panels menu button located in the Command window. A point may be copied to the Selected Point window, written to the standard output device, deleted, or simply examined.

**Panel items:**

- Memory object stack setting: Allows the user to choose a memory object for inspection. The memory objects are:

  **Continuation:** Points calculated during a continuation calculation, by either the Periodic Orbits or Equilibrium Continuation window.

  **Fixed:** Fixed points and stable and unstable manifolds.

  **Mult:** Trajectory data for multiple trajectories, *i.e.*, trajectories computed from the Multiple Orbits window.

  **Traj:** Trajectory data for single trajectories, *i.e.*, trajectories computed from the Orbits window. This is the default setting.

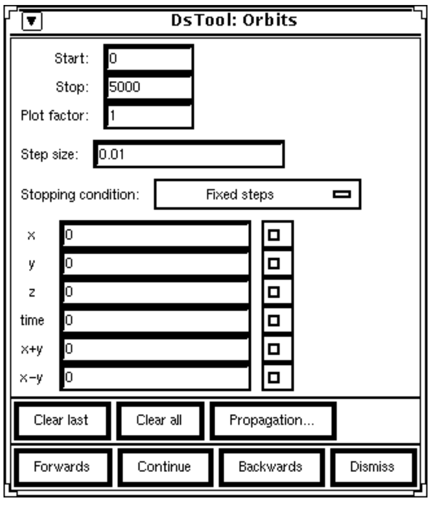
  **Param:** Points with specified parameters. Data will not be stored in this memory object by DsTool Tk, but may be loaded in from data files.

  **Sel_Pt:** Points saved by the user from the Selected Point window.

- Flow numeric field: Displays the number of the flow to investigate. Flows are numbered sequentially, with the earliest flow being number 1, and the latest flow being number $n$.
- Flow number increase command button: Increases the flow number by one.
- Flow number decrease command button: Decreases the flow number by one.
- Range of flows message: Displays the range of flows which are currently in memory.
- Trajectory numeric field: Displays the number of the trajectory to investigate.
- Trajectory number increase command button: Increases the trajectory number by one.

- Trajectory number decrease command button: Decreases the trajectory number by one.
- Range of trajectories message: Displays the range of trajectories in the specified flow.
- Point numeric field: Displays the point from the specified trajectory to investigate. Points are numbered sequentially.
- Point number increase command button: Increases the point number by one.
- Point number decrease command button: Decreases the point number by one.
- Range of points message: Displays the range of points in the specified trajectory of the specified flow.
- Highlight exclusive setting: Allows the user to choose wheher highlighting is turned on. Setting this field to "on" causes the point currently specified by the above fields to be highlighted in each 2-D View window. Highlighting consists of marking the position of the point by a large set of crosshairs. When an invalid point is specified, no highlight is done. The default is to have highlighting off.
- *Variablename* read-only text fields: Display the value of the variable or parameter for the specified point in memory. There is one read-write text field for each variable and parameter in the defined dynamical system. If the point specified by the flow, trajectory, and point fields is not a valid point in memory, then these fields are blank.
- Color read-only text fields: Display the color information for the point in memory. The first color is the alternating color, the second is the pick color, and the last is the symbol code. See section 3.3 on colors in DsTool for additional information.
- Copy command button: Copies the current point specified by the browser into the initial column of the Selected Point window.
- Output command button: Writes the variable, parameter, and color fields of the currently displayed point to the standard output device (normally the terminal).
- Delete command button: Deletes the selected flow. Dstool will **not** ask for confirmation. The stored points field on the Command window is adjusted, as are the range fields above.
- Dismiss command button: Closes the Browser window.

## 2.20 The Orbits Window


DsTool: Orbits

Start: 0
Stop: 5000
Plot factor: 1
Step size: 0.01
Stopping condition: Fixed steps

x 0
y 0
z 0
time 0
x+y 0
x−y 0

Figure 2.20: The Orbits window.

**Window title:** DsTool: Orbits

**Function:** The Orbits window allows the user to set conditions for orbit propagation.

**Description:** The Orbits window is opened by selecting the Orbits option from the Panels menu button located in the Command window. The options on this window are used to control the numerical algorithms used to compute trajectories.

**Panel items:**

- Start numeric field: Displays the number of points to compute before beginning to plot and store data. Setting this text field to a positive integer means that initial transient behavior will not be displayed. The default value is 0.
- Stop numeric field: Displays the maximum number of steps that will be taken in computing a trajectory. Depending upon the value of Stopping condition, this field can have different meanings. If Stopping condition is set to Fixed steps, then Stop is used to compute the total number of time steps taken by the propagation routine (see Plot factor field). If Stopping condition is set to Event stopping or Fixed time, then Stop is used to compute the *maximum* number of time steps which the propagation routine may take. If Stopping condition is set to Poincaré section, then Stop is used to compute the maximum number of iterates of the Poincaré map. We assume that the trajectory being computed does not propagate to infinity, and also that the propagator does not fail in any way. A failure may occur, for example, when trying to implicitly iterate a map backwards. The default value is 5000.
- Plot factor numeric field: Displays the number of points to skip between points stored and plotted. That is, the $i$th point calculated will be stored to memory and plotted only if $i$ is greater than $start$ and $i - start$ is evenly divisible by $plot$. The exception to this rule is that the initial and final conditions are always stored and plotted. This field is replaced by the Max steps between sections field when Stopping condition is set to Poincaré section. The default value is 1.

  Example: If the previous three text fields read Start=500, Stop=1000, and Plot=3, then DsTool will calculate 1000 points along a trajectory but will only store the points indexed by 500, 503, 506,..., 998, and 1000 (the final condition is *always* kept). Setting Plot $= k > 1$ for mappings may be thought of as studying the $k$th iterate of the map. For vector fields, setting Plot $> 1$ is useful when it is necessary to choose a small integration step, but we do not want to see every plotted point.
- Max steps between sections numeric field: Displays the maximum number of integration steps which the propagation routine may take between hitting Poincaré sections. This field takes the place of the Plot factor field when Stopping condition is set to Poincaré section. If a section is not crossed before taking this many steps, then the propagation routine stops and returns the last plotted point. The default value is 1.
- Step size read-write text field: Displays the initial step size with which to begin vector field integration. This field is only visible if the dynamical system is a vector field. The default value is 0.01. Like the Stop field, the interpretation of the value in the Step size field depends upon the value of the stack setting entitled Stopping condition and upon the current choice of the integrator.

  If the current integrator is a *fixed step* algorithm (*e.g.*, the Euler method or a standard fourth order Runge-Kutta method):
  - If the Stopping condition is Fixed steps, the size of each integration step is constant and has the value shown in the Step size field.
  - If the Stopping condition is Event stopping, the integration step is initially set to the the value shown in the Step size field, and no step size will ever be larger that this value. The given step

size is used by the integration algorithm until an event is detected. The final step (which may be smaller than the value of Step size) used by the integrator is chosen so that the last point of the trajectory satisfies the given condition even up to some tolerance.

– If the Stopping condition is Fixed time, a new time step is computed for use by the integration routine. The new step is less than or equal to the value shown in the Step Size field, but is the largest step which divides that flight time by an integral value.

If the current integrator is a *variable step* algorithm (*e.g.*, quality-control algorithms and the Bulirsch-Stoer method), then the value shown in the Step size field is used as an initial step size.

- Stopping condition stack setting: Allows the user to choose the type of event which will terminate a trajectory calculation. There are several options. The exact number depends on whether the current dynamical system is a mapping or a vector field and, if it is a vector field, on the currently installed integrator.

  **Fixed steps:** The propagation algorithm will attempt to propagate the trajectory for the number of time steps specified by the Stop read-write field. This is the default setting.

  **Event stopping:** The propagation algorithm will halt propagation as soon as *any* of the specified events are satisfied. An event is specified by typing in a value for an event field, and checking the non-exclusive field to the right of that field. Any variable or auxiliary function may be used to determine an event. For example, if we wish to propagate a trajectory until the scalar auxiliary function $g$ is zero, then we would type 0.0 into the event field next to the label for $g$ and check the non-exclusive field to the right of $g$. If we want to propagate until the variable $x$ reaches the value 0.5 *or* until the auxiliary function $g$ is 1.0, then we would type both of these values into the event fields labeled by $x$ and $g$ and check both corresponding non-exclusive fields. The propagator will stop the first time that *any* of the specified events are true. The parameters controlling the Newton iterations and tolerance for the event are set in the Propagation window.

  **Fixed time:** The propagation algorithm will attempt to propagate a trajectory until a final value of the independent variable is satisfied. This field only exists when the current dynamical system is a vector field. The label for this condition is always Fixed *indep_varb* where *indep_varb* is the name of the independent variable for the current vector field (often "time" is used). This is currently implemented by using Newton's method to solve for a final time step so that the last point satisfies the condition to within some error tolerance. The parameters for controlling Newton's method are set in the Propagation window.

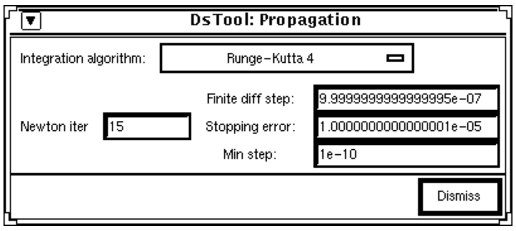
  **Poincaré section:** The propagation algorithm will set up a section or union of sections as stopping events in the same fashion as Event stopping. This field exists only when the current dynamical system is a vector field. The trajectory is flowed from the initial condition until the number of events specified by the Stop field have occurred. Each event starting with the one indicated by the Start field is plotted and saved. Intermediate points are neither plotted nor saved. The Max steps between sections field specifies the maximum number of propagation steps which the propagation routine is allowed to take between events before stopping.

- *Name* Event read-write text fields: Displays the values used to define events for stopping conditions. If the Stopping condition choice is Fixed steps, then none of these fields are relevant and trajectories will propagate as described above under the description of Start, Stop, and Plot factor. If the choice is Event stopping or Poincaré section then all dependent variables and auxiliary functions whose Event enable setting is on will be relevant in determining the stopping condition, as described above. If the choice is Fixed time then only the independent variable is relevant, and only then if its Event enable setting in on.

- *Name* Event enable non-exclusive settings: Allows the user to choose which events will be relevant to determining stopping conditions. Checking any of these settings causes the corresponding variable or auxiliary function to possibly become relevant in determining the stopping condition for the trajectory calculation. The relevance of the variable or auxiliary function is determined by the stopping condition field, as described above.

- Clear last command button: Deletes the last computed trajectory segment from memory and updates the number of trajectory data points.
- Clear all command button: Deletes all stored trajectory data points, including those generated by the Multiple Orbits window.
- Propagation window button: Opens and brings to the foreground the Propagation window.
- Forwards command button: Begins propagation (*i.e.*, integrating a vector field or iterating a map) forward in time using the initial condition given in the Initial column of the Selected Point window and the settings specified by the above fields.
- Continue command button: Begins propagation in the current direction, using the final point of the last computed trajectory as the initial condition and the settings specified by the above fields. If there is no last trajectory, selecting Continue is equivalent to selecting Forwards. The current direction is "forward" if the user selected the Forwards button or the SELECT mouse button more recently than the user selected the Backwards button or the MENU mouse button. (See Section 2.9 or 2.10 for more information on the effects of mouse buttons.)
- Backwards command button: Begins propagation backwards in time using the initial condition given in the Initial column of the Selected Point window and the settings specified by the above fields. If the installed dynamical system is a map, finding inverse images takes on two meanings. If the map has an explicit inverse, the inverse image is computed explicitly. If a map does not have an explicit inverse, Newton's method is used to determine approximate inverse images. The Jacobian of the map, the convergence criteria for Newton's method, and other relevant parameters for this numerical procedure may be set by using the Propagation window.
- Dismiss command button: Closes the Orbits window.

## 2.21 The Propagation Window



DsTool: Propagation

Integration algorithm: Runge-Kutta 4

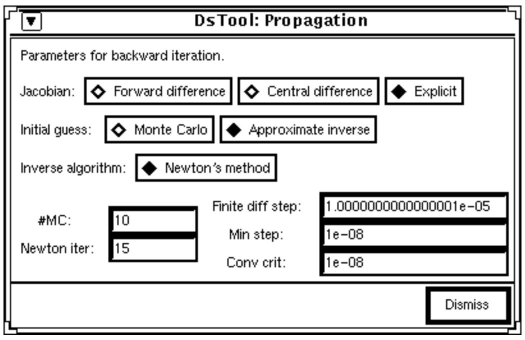
Finite diff step: 9.9999999999999995e-07

Newton iter 15

Stopping error: 1.0000000000000001e-05

Min step: 1e-10

Figure 2.21: A Propagation window for vector fields.



DsTool: Propagation

Parameters for backward iteration.

Jacobian: Forward difference | Central difference | [✓ Explicit]

Initial guess: Monte Carlo | [✓ Approximate inverse]

Inverse algorithm: [✓ Newton's method]

#MC: 10

Newton iter: 15

Finite diff step: 1.0000000000000001e-05

Min step: 1e-08

Conv crit: 1e-08

Figure 2.22: A Propagation window for mappings.

**Window title:** DsTool: Propagation

**Function:** The Propagation window allows the user to select and control the algorithms used to compute trajectories.

**Description:** The Propagation window is opened by selecting the Propagation window button located in the Orbits window. The Propagation window contains parameters necessary to control the forward and backward propagation of trajectories. The window's appearance depends upon the model chosen as well as the method of propagation.

**Panel items:** We first describe the Propagation window as it appears for vector fields:

The panel items in the Propagation window will change with the method of propagation selected. The top portion of the panel consists of $k \geq 1$ stack settings from which the user may select algorithms used by the propagator. The lower left section of the panel consists of $i \geq 0$ numeric fields which are used to control integer parameters of the algorithms chosen. These fields may be used, for example, to set the maximum number of Newton iterations used in estimating an inverse image for a diffeomorphism is one example of an integer parameter. The lower right section of the panel contains a list of $j \geq 0$ double precision fields which are used to control floating point parameters of the chosen numerical algorithms. These fields may be used, for example, to set the minimum step size allowable for variable-step integrators, or to establish convergence criteria for Newton's method.

- Integration algorithm stack setting: Allows the user to choose a numerical integrator. The current options are:

  **Runge-Kutta 4:** A fourth-order Runge-Kutta algorithm. This is the default.

  **Euler:** The forward Euler method.

  **Runge-Kutta 4QC:** A fourth-order Runge-Kutta algorithm with fifth-order stepsize regulation.

  **Runge-Kutta-Fehlberg 78:** A Runge-Kutta-Fehlberg algorithm of orders seven and eight.

  **Bulirsch-Stoer:** The Bulirsch-Stoer algorithm with Richardson extrapolation.

  **Adams-Bashforth 4:** The fourth-order Adams-Bashforth algorithm.

  See, for example, references [4, 5] for more information about these algorithms.

- Newton iter numeric field: Displays the maximum number of iterations allowed in determining the point along a trajectory for which some event is satisfied. The default value is 15.

- Finite diff step read-write text field: Displays the finite difference step size. When the Stopping condition stack on the Orbits window is set to Event stopping, the algorithm uses forward differencing to estimate the derivative of the stopping criterion function with respect to the time step. The default value is $10^{-6}$.

- Stopping error read-write text field: Displays the upper bound on the maximum error tolerated during event stopping. In other words, if the stopping condition is, say, $g(x) = c$ then the maximum allowable norm of $g(x) - c$ will be the value of this field. The default value is $10^{-5}$.

- Min dt read-write text field: Displays the smallest allowable time step. This value is important even for fixed-step integrators such as standard Runge-Kutta algorithms, because once a stopping event is detected, the time step is varied so that the trajectory satisfies the stopping condition to within the error specified by the Error field. The default value is $10^{-10}$.

  The remainder of the panel items are determined by which algorithm is selected. Therefore, for each integrator, we now describe the remainder of the panel.

- Runge-Kutta 4, Euler, Adams-Bashforth 4:

  No additional fields.

- Runge-Kutta 4QC:

  Const. fractional err. / Bounded global errors exclusive setting: Allows the user to choose between controlling the errors at each step by a constant amount, or controlling the errors globally. The default is to have constant fractional errors.

Max step read-write text field: Displays the value of the largest allowable time step. The default value is 0.2.

Err tol read-write text field: Displays the value of the required accuracy. The default value is $10^{-5}$.

Pgrow read-write text field: Displays the value of PGROW (see [4]) for the RK-4QC algorithm. The default value is $-0.2$.

Pshrink read-write text field: Displays the value of PSHRNK (see [4]) for the RK-4QC algorithm. The default value is $-0.25$.

Fcor read-write text field: Displays the value of FCOR (see [4]) for the RK-4QC algorithm. The default value is 0.6666.

Safety read-write text field: Displays the value of SAFETY (see [4]) for the RK-4QC algorithm. The default value is 0.9.

- Runge-Kutta-Fehlberg 78:

  Max step read-write text field: Displays the value of the largest allowable time step. The default value is 0.2.

  Err tol read-write text field: Displays the value of the required accuracy. The default value is $10^{-5}$.

  Safety read-write text field: Displays the factor to reduce the step size by ($\alpha$ in [5]) for the RKF algorithm. The default value is 0.8.

- Bulirsch-Stoer:

  Const. fractional err / Bounded global errors exclusive setting: Allows the user to choose between controlling the errors at each step by a constant amount, or controlling the errors globally. The default is to have constant fractional errors.

  BS intervals numeric field: Displays the integer which will be used by the Bulirsch-Stoer algorithm as the number of Bulirsch-Stoer intervals (NUSE in [4]). The default value is 7.

  Max step read-write text field: Displays the value of the largest allowable time step. The default value is 0.2.

  Err tol read-write text field: Displays the value of the required accuracy. The default value is $10^{-5}$.

  Shrink read-write text field: Displays the value of SHRINK (see [4]) for the B-S algorithm. The default value is 0.95.

  Grow read-write text field: Displays the value of GROW (see [4]) for the B-S algorithm. The default value is 1.2.

We now describe the Propagation window as it appears for mappings:

The panel entries of the Propagation window will depend upon the current dynamical system. In particular, an option will not be displayed if the current model does not allow the method in question to be used.

- Jacobian exclusive setting: Allows the user to choose how to evaluate the Jacobian matrix for the map. If the mapping does not have an explicit inverse, then Jacobian matrix is needed in order to compute inverse images of points, *i.e.*, for backwards iteration of the map. The options are:

  **Forward difference:** A numerical Jacobian will be used, calculated using a forward difference method. This method is $\mathcal{O}(h)$ in the finite difference step $h$.

  **Central difference:** A numerical Jacobian will be used, calculated using a central difference method. This method is $\mathcal{O}(h^2)$ in the finite difference step $h$.

**Explicit:** The explicit Jacobian will be used. This option is only available if the user has supplied an explicit Jacobian for the map.

- Initial guess exclusive setting: Allows the user to choose how to pick the initial guess ("seed") for Newton's method. This guess may be provided by:

  **Approx inv:** The seed is chosen from an approximate inverse. If the map may be considered a perturbation of a map which *does* have an exact inverse, a good guess for the inverse of the perturbed system is often given by the exact inverse of the unperturbed system. A few steps of Newton's method is often sufficient to converge to the inverse of the perturbed system. This option is not available if the mapping does not have an approximate inverse defined, *i.e.*, if the inverse_toggle model file variable is set to either FALSE or EXPLICIT_INV.

  **Monte Carlo:** The seed is chosen at random from within the hypervolume defined by the coordinate values in the Defaults window.

- Inverse algorithm exclusive setting: Allows the user to choose which type of inverse algorithm will be used to compute approximate inverse images. The options are:

  **Newton's method:** Newton's method is used to calculate pre-images.

  **Explicit formula:** An explicit formula is used to calculate pre-images. This option is not available if the mapping does not have an explicit inverse function defined, *i.e.*, if the inverse_toggle model file variable is set to either FALSE or APPROX_INV.

- #MC numeric field: Displays the maximum number of random guesses taken by the Monte Carlo routine. The default value is 10.

- Newton iter numeric field: Displays the maximum number of iterations allowed in Newton's method of computing fixed points. This algorithm is used, for example, in determining the point along a trajectory for which some event is satisfied. The default value is 15.

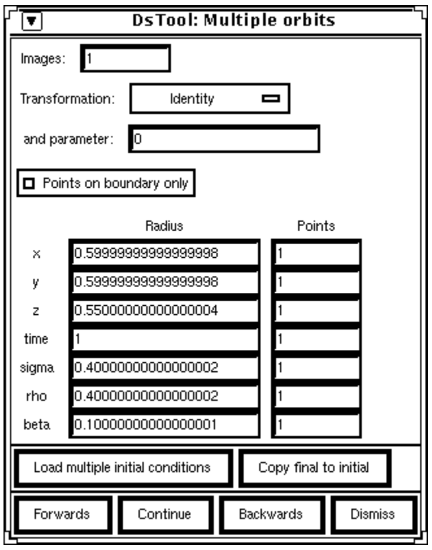
- Finite diff step read-write text field: Displays the spatial step to be used for computing a finite difference Jacobian. The default value is $10^{-5}$. This field is only used if the Jacobian exclusive setting is set to either Forward difference or Central difference.

- Min step read-write text field: Displays the minimum step required to take during Newton's method. Newton's method generates a sequence of points $\{x_i\}$ which (hopefully) converges to the inverse image. The difference between $x_{i+1}$ and $x_i$ is called the $i$th Newton step. If the length of the Newton step is less than the value of Min step, then we assume that we can no longer improve our current guess, and so we end the Newton process. The default value is $10^{-8}$.

- Conv crit read-write text field: Displays the criterion used to determine when Newton's method has converged. We use Newton's method to compute a root of some function, say, $g$. An iterative sequence $\{x_i\}_0^n$ is said to converge to a solution if the norm of $g(x_i)$ is less than the value of Conv crit for some $i$. The default value is $10^{-8}$.

- Dismiss command button: Closes the Propagation window.

## 2.22 The Multiple Orbits Window



Images: 1

Transformation: Identity

and parameter: 0

☐ Points on boundary only

| | Radius | Points |
|---|---|---|
| x | 0.59999999999999998 | 1 |
| y | 0.59999999999999998 | 1 |
| z | 0.55000000000000004 | 1 |
| time | 1 | 1 |
| sigma | 0.40000000000000002 | 1 |
| rho | 0.40000000000000002 | 1 |
| beta | 0.10000000000000001 | 1 |

Figure 2.23: The Multiple Orbits window.

**Window title:** DsTool: Multiple orbits

**Function:** The Multiple Orbits window allows the user to select a set of initial conditions from which to start trajectories.

**Description:** The Multiple Orbits window is opened by selecting the Multiple Orbits option from the Panels menu button located in the Command window. The user selects a region of the product of the phase space and parameter space centered about the point specified in the Selected Point window. The user may specify that points be chosen on a regular grid inside the region or only on the boundary. In the former case, a "box", including its interior, is selected. The dimension of the box depends on the number of coordinates which have Number of points specified greater than 1. The total number of points will be the product of the values in the Number of points fields. In the latter case, the points will form only the outline of a rectangle in two dimensions. The two dimensions selected will be the first two coordinates in the list with the Number of points specified greater than 1. If there is only one coordinate with Number of points greater than 1, then the a line segment is selected. If all the Number of points fields are set to 1, then only the point specified in the Selected Point window is selected. Each edge of the set of selected points is discretized by the Number of points specified for the associated coordinate, and has length twice the value of the Radius field for that coordinate.

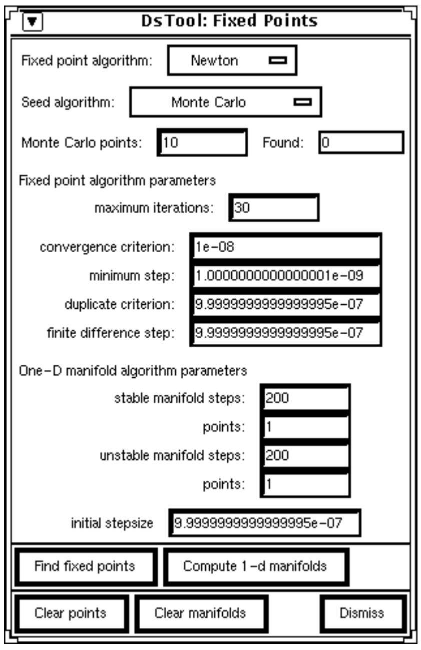
If a transformation other than Identity is selected, then each of these points will be passed through this filter. Transformations may be defined so that these points can be projected onto some submanifold of the whole space or to create arrays of points which are not in the form of a "box".

Initial multiple points may also be loaded by using the mouse interface. Dragging the SELECT button while holding down CONTROL in a 2-D View window will draw a rectangle in the viewing canvas. Upon release of the mouse button, a notice will appear allowing the user two choices. The selections are Load IC's and Cancel. If the user selects Load IC's, the appropriate Radius fields on the Multiple Orbits window are changed, and the new initial conditions are loaded. However, no orbits are computed. If the user selects Cancel, no action is taken.

**Panel items:**

- Images numeric field: Displays the number of automatic continuations to be performed. If $n$ is the value of this field, then selecting Forwards or Backwards with $n > 1$ will produce the same result as selecting Forwards or Backwards with $n = 1$ and then selecting Continue $n - 1$ times. The default value is 1.
- Transformation stack setting: Allows the user to choose a transformation to apply to the selected set of points. If this is set to anything but the identity, then whenever any points are loaded they will automatically be mapped through the transformation. The transformation functions must be defined and installed ahead of time. The options are Identity (the default), Elliptical, and Rotated.
- and parameter read-write text field: Displays the parameter to pass to the transformation function. This value will have different effects (or none at all) depending on the selected function. For the identity function, it has no effect. The default value is 0.
- Points on boundary only exclusive setting: Allows the user to choose whether the shape of the region to be selected is a "box" with interior or the outline of a rectangle (only the boundary of the "box"). The default is off, so that the shape of the region selected is a box.
- Radius read-write text fields: Display the radius of the box in the direction of each variable and parameter. Note that the side of the box will have twice this length.
- Points numeric fields: Display the the number of grid points to put along the specified side or axis of the box.
- Load multiple initial conditions command button: Loads the set of initial conditions using the current values on the Multiple Orbits and Selected Point windows.
- Copy final to initial command button: Loads the final values of the trajectories computed from the current initial conditions.
- Forwards command button: Starts the computation of forward trajectories from each initial condition. If there are no points loaded then points will be loaded as if the user had first pressed the Load multiple initial conditions command button. Trajectories are computed according to the specifications on the Orbits window.
- Continue command button: Extends the trajectories computed by the Forwards or Backwards command buttons on the Multiple Orbits window.
- Backwards command button: Starts the computation of backward trajectories from each initial condition. If there are no points loaded then points will be loaded as if the user had first pressed the Load multiple initial conditions command button. Trajectories are computed according to the specifications on the Orbits window.
- Dismiss command button: Closes the Multiple Orbits window.

## 2.23 The Fixed Points Window



DsTool: Fixed Points

Fixed point algorithm: Newton

Seed algorithm: Monte Carlo

Monte Carlo points: 10 Found: 0

Fixed point algorithm parameters

maximum iterations: 30

convergence criterion: 1e-08

minimum step: 1.0000000000000001e-09

duplicate criterion: 9.9999999999999995e-07

finite difference step: 9.9999999999999995e-07

One-D manifold algorithm parameters

stable manifold steps: 200

points: 1

unstable manifold steps: 200

points: 1

initial stepsize 9.9999999999999995e-07

Figure 2.24: The Fixed Points window.

**Window title:** DsTool: Fixed Points

**Function:** The Fixed Points window allows the user to find equilibria and to compute one dimensional stable and unstable manifolds.

**Description:** The Fixed Points window is opened by selecting the Fixed Points option from the Panels menu button located in the Command window. The window contains parameters for controlling the computation of stable and unstable manifolds.

Color coding is performed according to the stability type of objects: saddle points are marked as green crosses, sinks are blue triangles, and sources are red squares. Furthermore, unstable manifolds are plotted in red and stable manifolds are plotted in blue.

**Panel items:**

- Fixed point algorithm stack setting: Allows the user to choose an algorithm to use for root-finding. The current options are the Secant method and Newton's method. The default is Newton's method.
- Seed algorithm stack setting: Allows the user to choose the method of providing an initial guess ("seed") to the root-finding algorithm. The choices are:

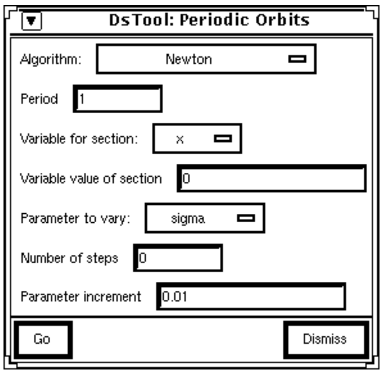
  **Monte Carlo:** The seed is a pseudo-random point chosen from the volume in phase space defined by the coordinate values in the Defaults window. This is the default setting.

  **Selected Point:** The seed is the currently selected point. The currently selected point is displayed in the Selected Point window.
- Monte Carlo points numeric field: Displays the number of guesses taken by the Monte Carlo routine. If the Seed algorithm setting is Selected Point, then this field is not relevant. The default value is 10.

- Period numeric field: Displays the integer length of periodic orbits to search for. This field is only shown if the current dynamical system is a mapping. Note that points with lower periods which divide the specified period may also be located. The default value is 1. Setting this field to a positive number should be used to search for periodic points (for mappings), instead of using the Periodic Orbits window.
- Found read-only text field: Displays the number of periodic orbits or equilibrium points found by DsTool's fixed point algorithms. This may be different than the total number of fixed points stored in memory, since fixed points may be loaded into memory from a data file. For a mapping, this field displays the number of *orbits* found, not the total number of points found.
- Fixed point algorithm parameters message: Identifies the text items which follow as parameters associated with the finding of fixed points.
- maximum iterations numeric field: Displays the maximum number of iterations allowed while trying to converge to a periodic point. The default value is 30.
- convergence criterion read-write text field: Displays the criterion used to determine when Newton's method has converged. An iterative sequence is said to converge to a periodic point (resp. equilibrium) if the norm of $f(x_i) - x_i$ (resp. $f(x_i)$) is less than the value of this field. Here $f$ is some iterate of the mapping (resp. $f$ defines the vector field). The default value is $10^{-8}$.
- minimum step read-write text field: Displays the minimum step to take in Newton's method. Periodic points (resp. equilibria) are found via an iterative process. If the difference between successive points is less than the value in this field, then no further steps will be taken and the final iterative point will not be regarded as a periodic point. The default value is $10^{-8}$.
- duplicate criterion read-write text field: Displays the distance used to determine when two points should be considered the same. Two periodic points (resp. equilibria) are considered to be identical if the norm of their difference is less than the entry in this field. The default value is $10^{-6}$.
- finite difference step read-write text field: Displays the stepsize to use when using finite differencing in the DsTool fixed point calculations. The default value is $10^{-6}$.
- One-D manifold algorithm parameters message: Identifies the text items which follow as parameters associated with the calculation of one dimensional stable and unstable manifolds.
- stable manifold steps numeric field: Displays the number of time steps to take in computing the manifold. The length of the stable manifold displayed depends on the strength of the eigenvalues and the number of time steps taken along the manifolds. The default value is 200. This number should be increased for manifolds associated with small eigenvalues.
- (stable manifold) points numeric field: Displays the number of interpolating points between steps. Given a stable manifold and its pre-image, the manifold may be better visualized by interpolating additional points along the manifold. The default value is 1, but for manifolds which oscillate wildly (*e.g.*, the transverse intersection of homoclinic orbits) this number can be chosen much larger.
- unstable manifold steps numeric field: Displays the number of time steps to take in computing the manifold. The length of the unstable manifold displayed depends on the strength of the eigenvalues and the number of time steps taken along the manifolds. The default value is 200. This number should be increased for manifolds associated with large eigenvalues.
- (unstable manifold) points numeric field: Displays the number of interpolating points between steps. Given an unstable manifold and its pre-image, the manifold may be better visualized by interpolating additional points along the manifold. The default value is 1, but for manifolds which oscillate wildly (*e.g.*, the transverse intersection of homoclinic orbits) this number can be chosen much larger.

- Initial stepsize read-write text field: Displays the distance from the periodic point (resp. equilibrium) along the eigenvector to choose the initial point of the stable or unstable manifold calculation. The default value is $10^{-6}$. This number should be decreased for highly nonlinear systems, *i.e.*, when the linearization at a periodic point (resp. equilibria) is only a good approximation to the (nonlinear) mapping (resp. vector field) within a tiny neighborhood of the periodic point (resp. equilibria).
- Find fixed points command button: Initiates the search for equilibria, using the current window settings.
- Compute 1-d manifolds command button: Initiates the calculation and display of one dimensional stable and unstable manifolds.
- Clear points command button: Erases the periodic points in memory and resets the Found text field to 0.
- Clear manifolds command button: Erases the manifolds computed by the Fixed Points window.
- Dismiss command button: Closes the Fixed Points window.

## 2.24 The Periodic Orbits Window


DsTool: Periodic Orbits

Algorithm: Newton

Period 1

Variable for section: x

Variable value of section 0

Parameter to vary: sigma

Number of steps 0

Parameter increment 0.01

Figure 2.25: The Periodic Orbits window.

**Window title:** DsTool: Periodic Orbits

**Function:** The Periodic Orbits window allows the user to find periodic orbits and to do continuation calculations for periodic orbits of vector fields.

**Description:** The Periodic Orbits window is opened by selecting the Periodic Orbits option from the Panels menu button located in the Command window. The window contains parameters for controlling the finding of periodic orbits and doing continuation calculations. This window should not be used for mappings. Instead, the Fixed Points window should be used, with its Period field set to a positive integer.

Colorcoding is performed according to the stability type of objects: saddle orbits are colored green, attracting orbits are blue, and repelling orbits are red.

**Panel items:**

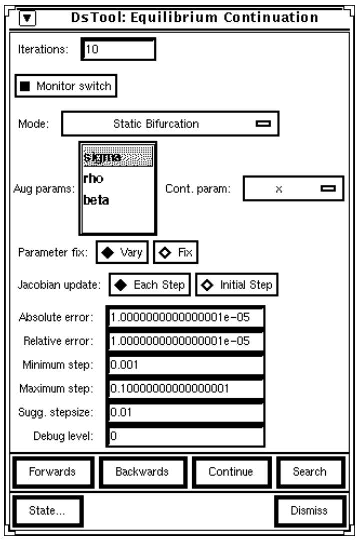
- Algorithm stack setting: Allows the user to choose which algorithm will be used to find periodic orbits. The current options are:

**Newton:** Use Newton's method applied to the Poincaré map defined by the subsequent variables. Derivative calculations are done by finite differencing. This is the default option.

**Attracting PO only:** Use the fact that attracting periodic orbits attract nearby trajectories to find the periodic orbit. That is, we integrate forward in time until either we've come close enough to the periodic orbit (*i.e.*, the difference between subsequent iterates of the Poincaré map is less than the convergence criterion defined in the Fixed Points window), or we give up (reach the value of Stop in the Orbits window).

- Period numeric field: Displays the period of the fixed point of the Poincaré map to search for. In most cases, having this field set to the default value (which is 1) will be adequate.
- Variable for section stack setting: Allows the user to choose the variable which is used to create the Poincaré section. Allowable options are the dependent variables of the current dynamical system.
- Variable value of section read-write text field: Displays the value of the section variable which is used to create the Poincaré section. That is, if the Variable for section is "x", and this field is set to "a", then the Poincaré section is defined as the $n-1$ dimensional hyperplane $x = a$.
- Parameter to vary stack setting: Allows the user to choose a parameter to perform continuation calculations with. Allowable options are the parameters of the current dynamical system.
- Number of steps numeric field: Displays the (maximum) number of steps of size parameter increment to take in the parameter specified by the Parameter to vary stack. For each step, we attempt to find a periodic orbit. If we fail to find a periodic orbit for a parameter value, the calculation ceases. Otherwise, we increment the parameter by the value of parameter increment and continue. If this field is set to 0 (the default value), the routines search for periodic orbits with the given parameter values.
- Parameter increment read-write text field: Displays the size of the increment of the specified parameter at each step of the continuation calculation. The default value is 0.01.
- Go command button: Initiates the periodic orbit (continuation) calculation, using the current values of the displayed fields.
- Dismiss command button: Closes the Periodic Orbits window.

## 2.25 The Equilibrium Continuation Window


DsTool: Equilibrium Continuation

Iterations: 10

■ Monitor switch

Mode: Static Bifurcation

Aug params: sigma, rho, beta

Cont. param: x

Parameter fix: [✓ Vary] Fix

Jacobian update: [✓ Each Step] Initial Step

Absolute error: 1.0000000000000001e-05

Relative error: 1.0000000000000001e-05

Minimum step: 0.001

Maximum step: 0.10000000000000001

Sugg. stepsize: 0.01

Debug level: 0

Figure 2.26: The Continuation window.

**Window title:** DsTool: Equilibrium Continuation

**Function:** The Equilibrium Continuation window allows the user to compute curves of bifurcations.

**Description:** The Equilibrium Continuation window is opened by selecting the Equilibrium Continuation option from the Panels menu button located in the Command window. The window allows the user to compute curves of equilibrium points with one varying parameter and curves of saddle-node and Hopf bifurcations with two varying parameters. A brief tutorial on continuation calculations is at the end of this section. The different Hopf bifurcation algorithms are described in [2] and [3].

Color coding for hyperbolic equlibrium points is performed according to the value of the Monitor switch setting. For the bifurcation types, color coding is performed according to the choice of colormap, as shown in the Continuation Colors window. In the default colormap, Hopf bifurcation points are displayed in magenta, saddle-node bifurcation points are displayed in green, degenerate Hopf points are displayed in orange, and resonant saddle-node points are displayed in sea green.

**Panel items:**

- Iterations numeric field: Displays the number of points along a curve of equilibrium points that will be computed. The type of curve that will be computed is determined by the Mode stack setting. The default value is 10.
- Monitor switch exclusive setting: Allows the user to choose whether or not to color equilibrium points with the DsTool convention: blue for sink, green for saddle and red for source. The default value is on.
- Mode stack setting: Allows the user to choose the system of equations to be solved with continuation. The choices are:

  **Static Bifurcation:** for continuing curves of equilibrium points with a single active parameter.

  **Saddle Node ( $\det() = 0$ ) :** for computing curves of saddle-node bifurcations with two active parameters. The determinant of the Jacobian is computed and used as an augmenting equation to the equilibrium equations.

**Saddle Node Bifurcation:** for computing curves of saddle-node bifurcations with two active parameters.

**Hopf Bif ( $|Bp = 0|$ ) :** for computing curves of Hopf bifurcations with two active parameters, using the bialternate product.

**Hopf Bif ( $|\text{Bezout}| = 0$ ) :** for computing curves of Hopf bifurcations with two active parameters, using the Bezoutian.

**Hopf Bif (JGR):** for computing curves of Hopf bifurcations with two active parameters, using the algorithm developed by Jepson, Griewank and Reddien.

**Hopf Bif (kubicek 1):** for computing curves of Hopf bifurcations with two active parameters, using the algorithm developed by Kubicek.

- Aug params listbox: Allows the user to choose active parameters. Static Bifurcation calculations require that the user select one active parameter. Saddle-node and Hopf bifurcation calculations require the user to select two active parameters. An error message will be printed if the user tries to select too many active parameters, or tries to initiate a calculation with too few active parameters.
- Cont. param stack setting: Allows the user to choose one of the phase space variables or one of the active parameters to parameterize the curve of equilibrium points.
- Parameter fix switch exclusive setting: Allows the user to choose whether the setting in Cont. param stack may be changed during the calculation. The choice Vary allows the continuation algorithm to change the variable used as the continuation parameter adaptively. The choice Fix uses requires the continuation algorithm to use only the variable selected in the Cont. param stack setting. The default is the "Vary" setting.
- Jacobian update switch exclusive setting: Allows the user to choose to update the Jacobian of the augmented system at each step or only at the initial continuation step. The default is to update the Jacobian at each step.
- Absolute error read-write text field: Displays the absolute error tolerance used in solving the augmented equations. The default value is $10^{-5}$.
- Relative error read-write text field: Displays the relative error tolerance used in solving the augmented equations. The default value is $10^{-5}$.
- Minimum step read-write text field: Displays the minimum step size used by the continuation algorithm for continuation steps of the continuation parameter. The default value is $10^{-3}$.
- Maximum step read-write text field: Displays the maximum step size used by the continuation algorithm for continuation steps of the continuation parameter. The default value is 0.1.
- Sugg. step read-write text field: Displays the step size used by the continuation algorithm for the initial continuation step of the continuation parameter. The default value is 0.01.
- Debug level numeric field: Displays an integer for determining the amount of diagnostic information printed to the terminal window from which DsTool is run. The default value is 0, which prints no diagnostic information to the terminal.
- Forwards command button: Initiates a continuation calculation in the direction of increasing values of the continuation parameter for the number of steps in the Iterations field.
- Backwards command button: Initiates a continuation calculation in the direction of decreasing values of the continuation parameter for the number of steps in the Iterations field.
- Continue command button: Continues the last continuation calculation for the number of steps in the Iterations field.
- Search command button: Uses Monte Carlo seeds to locate solutions of the augmented equations. The number of seeds used is determiend by the value of the Iterations text field.

- State window button: Opens and brings to the foreground the Continuation State window.
- Dismiss command button: Closes the Equilibrium Continuation window.

**Appendix: Continuation Calculations**

One parameter continuation is a systematic strategy for computing curves of solutions to $l$ equations $G : \mathbb{R}^{l+1} \to \mathbb{R}^l$ in $l+1$ variables. The mathematical foundation for continuation algorithms is the implicit function theorem. If $G$ is differentiable and the Jacobian of $G$ is surjective at a point $x \in \mathbb{R}^{l+1}$ where $G$ vanishes, then the solutions form a curve in a neighborhood of $x$. Moreover, the implicit function theorem gives a formula for the tangent vector to the solution curve. Continuation algorithms exploit this information. They use an initial value solver for ordinary differential equations (often just the Euler method) to step along an approximation to the solution curve. They then appply a root finding algorithm to significantly improve the approximation. The alternation of root finding with numerical integration steps distinguish continuation methods. Choices of how to parametrize the solution curve, choose time steps and restrict the equations to a hyperplane to obtain a square system of equations with a unique solution (locally) need to be made. Various continuation packages take different approaches to these matters: the continuation "engine" used by DsTool is PITCON (Rheinboldt ...).

If $f : \mathbb{R}^{n+k} \to \mathbb{R}^n$ is a $k$ parameter family of vector fields, then we assemble systems of equations $G$ for varied calculations. To do so we restrict ourselves to a submanifold of dimension $l+1 \subset \mathbb{R}^{n+k}$. Most frequently this submanifold is obtained by fixing $k-s$ of the parameters of $f$ (called **inactive parameters**) and varying $s$ **active** parameters. In the simplest case, $G = f$, the number of equations is $n$, $s = 1$ and the continuation calculation computes curves of equilibria with a single active parameter. To compute curves of bifurcations, one adds **defining equations** to the system of equations $f = 0$ to produce an **augmented** system. The number of independent defining equations added to $f = 0$ is the **codimension** of the bifurcation. (In some circumstances, the augmented system uses additional **auxiliary** variables such as eigenvalues of the Jacobian and has a corresponding number of additional equations.)

The continuation window of DsTool includes calculations for saddle-node and Hopf bifurcations. These are the codimension one bifurcations of equilibrium points. For saddle-node bifurcations, the defining equation is $\det(Df_x) = 0$ or an algorithm that computes another scalar quantity, such as the smallest singular value, that vanishes precisely when $Df$ is singular. For Hopf bifurcations, the defining equation(s) compute where the matrix $Df$ has a pair of pure imaginary eigenvalues. Explicit expressions in the coefficients of $Df$ that compute whether $Df$ has a pair of complex eigenvalues are very complicated. Alternate ways of performing this computation are discussed in [3]. There are also other algorithms that perform the calculation by introducing auxiliary variables for the pure imaginary eigenvalues, and in some cases the eigenvectors of $Df$.

## 2.26 The Continuation State Window

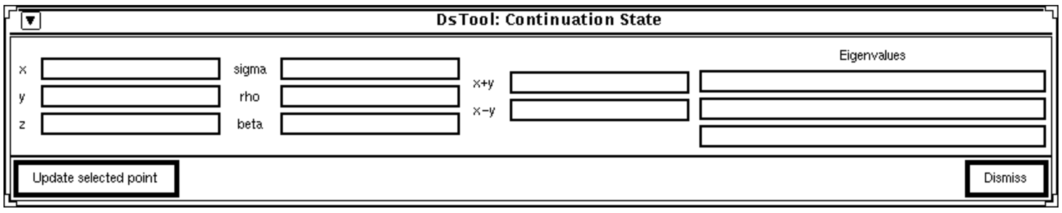


Figure 2.27: The Continuation State window.

**Window title:** DsTool: Continuation State

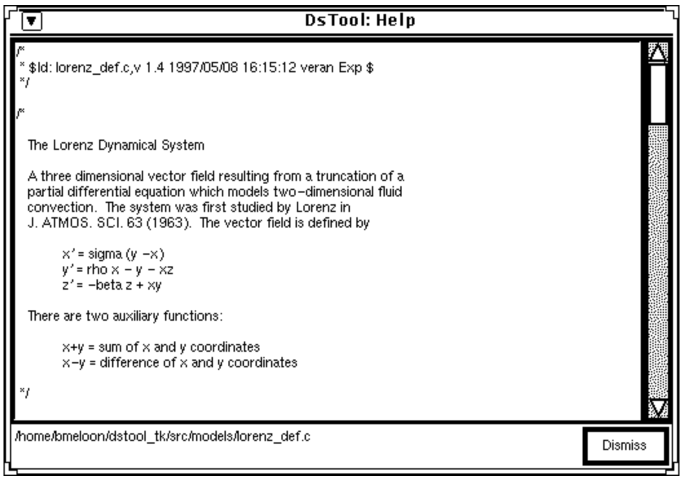
**Function:** The Continuation State window displays information about the last point computed by the continuation routines.

**Description:** The Continuation State window is opened by selecting the State window button in the Equilibrium Continuation window. The window allows the user to see the values of the variables, parameters, and auxiliary functions, as well as the eigenvalues of the Jacobian matrix for the last equilibrium point computed by the continuation routines. In addition, the point displayed can be copied to the Selected Point window.

**Panel items:**

- *Name* read-only text fields: Display the value of each variable, parameter, and auxiliary function in the current dynamical system at the end of the last computed continuation curve.
- Eigenvalues read-only text fields: Display the values of the eigenvalues of the Jacobian matrix at the last computed fixed point.
- Update selected point command button: Copies the displayed point into the Selected Point window.
- Dismiss command button: Closes the Continuation State window.

## 2.27 Help Windows



```
/*
 * $Id: lorenz_def.c,v 1.4 1997/05/08 16:15:12 veran Exp $
 */

/*

  The Lorenz Dynamical System

  A three dimensional vector field resulting from a truncation of a
  partial differential equation which models two-dimensional fluid
  convection.  The system was first studied by Lorenz in
  J. ATMOS. SCI. 63 (1963).  The vector field is defined by

        x' = sigma (y -x)
        y' = rho x - y - xz
        z' = -beta z + xy

  There are two auxiliary functions:

        x+y = sum of x and y coordinates
        x-y = difference of x and y coordinates

*/
```

/home/bmeloon/dstool_tk/src/models/lorenz_def.c

Figure 2.28: The Model Help window

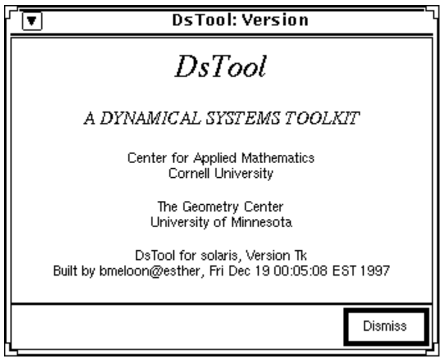
**Window Title:** DsTool: Help

**Function:** The Model Help window displays the C code for the current dynamical system.

**Description:** The Model Help window is opened by selecting the Model option from the Help menu button located in the Command window. It displays the C code which was is used to define the current dynamical system. If this file cannot be found, an error message is printed.

**Panel items:**

- Model Definition read-only text pane: Displays the model file (C code) used by DsTool to create the current model. See section 4.2 on defining dynamical systems for more information on what the various parts of the model file mean.
- File identifier message: Displays the location of the file which appears in the text pane.
- Dismiss command button: Closes the Model Help window.



DsTool: Version

*DsTool*

*A DYNAMICAL SYSTEMS TOOLKIT*

Center for Applied Mathematics
Cornell University

The Geometry Center
University of Minnesota

DsTool for solaris, Version Tk
Built by bmeloon@esther, Fri Dec 19 00:05:08 EST 1997

Figure 2.29: The About DsTool Help window

**Window title:** DsTool: Version

**Function:** The About DsTool Help window displays information about the current version of DsTool.

**Description:** The About DsTool Help window is opened by selecting the About DsTool option from the Help menu button located in the Command window.

**Panel items:**

- Information message: Displays information about the current executable. It includes information about which architecture this program was built for, when the current version was built and by whom, as well as information about the affiliated institutions of the authors.
- Dismiss command button: Closes the About DsTool Help window.

# Chapter 3

# Additional Topics

## 3.1 Defining Dynamical Systems Using the Parser Window

The parser allows the user to define a vector field or mapping in a simple textual format as opposed to C code language. This may be extended to a full description of the dynamical system by defining initial conditions and default ranges for the variables and parameters, declaring variables periodic on a specified interval, and defining auxiliary functions. Once a dynamical system has been defined in a text window, it may be interpreted and examined within DsTool.

The capabilities of the parser will be described by considering a number of examples. These examples are included as text files in the directory `$DSTOOL/data/parser/`, and may be loaded into the parser editor window.

While the parser is intended to be an easy-to-use general purpose tool, for extensive and/or more sophisticated calculations with a vector field or mapping, a full C-code model file should be compiled into DsTool. See section 4.2 for more information on this process. This is primarily because:

1. Computations with the parsed dynamical system will incur additional overhead since the code for evaluation of the dynamical system is not compiled.
2. Certain features cannot be defined by the parser. In addition, they will not be included into the automatically generated C code, and have to be added by hand. See section 3.1.8 on limitations for more details.

### 3.1.1 Basic Functionality

The basic defining equations for a dynamical system are entered in the following format:

```
# The Lorenz system (lorenz.def)
x' = sigma ( y - x )
y' = rho x - y - x z
z' = -beta z + x y

INITIAL sigma 10 rho 28 beta 2.6667
```

We denote the time derivative of a variable by a prime, `'`. This is followed by an equals sign and then the right hand side. A mapping is input in the same format using the prime to indicate the new point in phase space. Variable and parameter names are alphanumeric strings beginning with an alphabetic character. Any undefined string is understood to be a parameter in the system.

Comments are allowed at the end of any line and must begin with the pound character, #. As seen in the above example, multiplication is understood, when no arithmetic operator is present, but it may be explicitly indicated with an asterisk, *. The INITIAL declaration assigns initial values to the parameters.

The parser understands the following fundamental constants, operators, and special functions:

```
constants: PI, pi, E
operators: +, -, *, /, % (mod), ^
special functions: sin, cos, tan, asin, acos, atan, sinh, cosh,
        tanh, log, ln, exp, abs, sqrt
```

The special functions all correspond to standard C math library functions.

If you try this example, set the dynamical system type setting to Vector field, and then select the “Build model” button on the Parser window in order to create the dynamical system from the text.

### 3.1.2 Periodic Variables

Phase space variables may be declared periodic on a fixed interval by using the PERIODIC command. We define the following map on the torus, $T^2 = \mathbb{R}^2/\mathbb{Z}^2$:

```
# The Kim-Ostlund torus map (kotorus.def)
x’ = x + w1 - a / (2 PI) sin(2 PI y)
y’ = y + w2 - a / (2 PI) sin(2 PI x)

INITIAL w1 0.6 w2 0.805 a 0.7
PERIODIC x 0 1 y 0 1
```

The PERIODIC command should only be given once in the definition and all periodic variables should be declared, but optionally may be placed on separate lines. When trying this example, don’t forget to set the dynamical system type setting to Mapping before selecting “Build model”.

### 3.1.3 Initial Conditions and Default Ranges

Initial conditions for the parameters and variables and their default ranges may be declared using the INITIAL and RANGE commands. These specify specific initial values for the parameters and variables and the default ranges for display purposes. Any or all of the parameters and variables may be specified in either case. We look at Duffing’s equation, for example:

```
# The Duffing oscillator (duffing.def)
x’ = y
y’ = - alpha y + (beta - x^2) x + gamma cos(omega t)
t’ = 1

INITIAL alpha 0.5 beta 0 gamma 10 omega 1
        x -5 y 4 t 0
RANGE x -6 6 y -6 6
```

Note that in this example, we explicitly augment the phase space with the variable t (else the parser will think t is a parameter). When initial conditions or ranges are not specified, the default values as given in Table 3.1 are used.

### 3.1.4 Auxiliary Functions

Auxiliary functions may be declared by defining them *after* the dynamical system equations. They should have the same format as the dynamical system, but do not use the prime after the name of the auxiliary function. RANGE may not be specified for auxiliary functions. For example, in the Van der Pol system:

```
# The Van der Pol oscillator (vanderpol.def)
x' = y
y' = alpha (1 - x^2) y - x + beta cos(omega t)
t' = 1

Rsqr = x x + y y     # square of polar radial coordinate

INITIAL alpha 1.0 beta 0 omega 1.0
RANGE x 3 -3 y 3 -3
```

### 3.1.5 Temporary Functions

In order to improve evaluation efficiency, it is often useful to define temporary variables to be used in the definition of vector fields or auxiliary functions. This can be done by defining a function *before* defining the dynamical system. Here is an example where temporary variables are used in both the vector field definition and in the auxiliary functions definition.

```
# A D4 symmetric system (d4symm.def)

xsq = x x       # xsq and zsq are temporary functions
zsq = z z

x' = y
y' = x (mu - (xsq+zsq)) + delta x zsq +
     epsilon ((xsq+zsq) y + nu y + Axzw x (x y + z w) + Ayz2 y zsq
z' = w
w' = z (mu - (xsq+zsq)) + delta z xsq +
     epsilon ((xsq+zsq) w + nu w + Axzw z (x y + z w) + Ayz2 w xsq

Energy = 0.5 (y y + w w - mu (xsq + zsq) + 0.5 (xsq + zsq) (xsq + zsq) -
              delta xsq zsq)
AngMom = y z - x w

INITIAL mu 2 delta 0.95 epsilon 0 nu -3.52 Axzw 1 Ayz2 0
RANGE x -5 5 y -5 5 z -5 5 w -5 5
```

### 3.1.6 Automatic C Code Generation

For complicated dynamical systems or analyses which will require a large amount of computation, the user will wish to code the dynamical system in C. If the model is written for the parser, then the parser may be able to do most of the work of writing the C language procedures necessary to interface into DsTool. The system can be built and tested using the parser and when the user has decided on a set of satisfactory initial conditions and ranges, the user may fill in the Name field with a single descriptive word and select the Write C code command/window button. This will generate C code for the system, and open the C Code window.

The code may be immediately edited, if for example the user wishes to add explicit Jacobian or inverse routines. Additionally, the C code may be made more efficient or functions may be implemented which are unavailable in the parser. Any information which was not supplied through the parser will be assigned a default value. These are listed in the table in the following subsection. When the user is finished making any modifications, the C code file may be saved and compiled into DsTool in the standard way. Section 4.2 contains a complete description of this process.

### 3.1.7 Default Values

Table 3.1 gives the defaults which are used in defining the dynamical systems. The items in **bold** are ones which may be specified explicitly by the user within the textual description of the dynamical system.

| Item | Vector Field | Mapping |
|---|---|---|
| **Variable Initial Value** | 0.0 | 0.0 |
| **Variable Range Minimum** | -10.0 | -1.0 |
| **Variable Range Maximum** | 10.0 | 1.0 |
| Independent Variable Name | time | iter |
| Independent Variable Initial Value | 0.0 | 0 |
| Independent Variable Range Minimum | 0.0 | 0 |
| Independent Variable Range Maximum | 10000.0 | 10000 |
| **Parameter Initial Value** | 1.0 | 1.0 |
| **Parameter Range Minimum** | -10.0 | -10.0 |
| **Parameter Range Maximum** | 10.0 | 10.0 |
| Auxiliary Function Range Minimum | -10.0 | -10.0 |
| Auxiliary Function Range Maximum | 10.0 | 10.0 |

Table 3.1: Default values.

### 3.1.8 Limitations and Bugs

- The conversion to C code is weak. Essentially, the input must be written in C code format, with or without the multiplication operator. The exponentiation operator, ^, will be translated directly, but this is not a valid C command. Parentheses need to be included with the special operators for a correct translation to C.

- Ranges for auxiliary functions may not be set.

- The independent variable name, initial value, and default range cannot be set. Nonautonomous vector fields are not allowed; they must be embedded in autonomous ones by augmenting the phase space.

- The same temporary functions are used for both the definition of the dynamical system and the auxiliary functions.

- The parser cannot be used to define analytic Jacobians. Nor can it define explicit or approximate inverses. These need to be added by hand into the automatically generated C code.

## 3.2 Hints and Suggestions for Using the Interfaces

### 3.2.1 Tips for Using Geomview

In order to use DsTool Tk's three dimensional viewing capabilities, you need to have access to Geomview. For information on using Geomview, see the documentation accompanying it. This section gives some tips which are specific to Geomview's interaction with DsTool.

In order to use Geomview effectively with DsTool, you need to know how DsTool interacts with Geomview. Geomview is used for two purposes in DsTool: plotting and viewing three dimensional data, and doing animations. For three dimensional viewing, data previously computed by DsTool is sent to Geomview, where the user can view and manipulate the data. For animations, the data is sent to Geomview as it is computed, so that the animation can take place in real time.

Here are some other helpful tips:

1. You can scale the individual axes of a target sent to Geomview from DsTool using the external module "Transformer". This module allows you to scale axes individually, or in groups. This feature comes in handy when you have data that has widely varying scales. While DsTool automatically compensates for these cases, Geomview does not.
2. There are two basic types of animations that are done with Geomview. The first type involves sending transformation matrices from DsTool to Geomview at each time step of an integration to change the position of the objects. The second type involves replacing the objects in Geomview with new ones at each time step. Due to the large amounts of data being passed from DsTool to Geomview, animations can be very slow, even for the first type. Therefore, when doing animations, you might want to lower the number of time steps taken or choose a faster integrator.
3. You can delete trajectories or other data objects by using the "Delete" function (on the edit menu), but this option will delete only the target which is highlighted in the target menu. This target is also listed in the Target field of the Tools window. Selecting Delete when the target is "World" will delete everything. Deleting selected targets can be useful when for example you want to view the (un)stable manifold of a single fixed point. Since DsTool sends the computed stable and unstable manifolds of the fixed points individually, you can delete the ones you aren't interested in seeing. Alternately, you can delete the manifolds in DsTool before sending them to Geomview.
4. You can overlay plots of variables by using the 3-D View window and Geomview. For example, if you want to overlay plots of x vs. t, y vs. t and z vs. t, you can do this by the following procedure: First, send to Geomview (via the 3-D View window) the plot x vs. t vs. 0. Then copy the resulting Geomview target by selecting the Copy option from the Edit menu button. Next, send to Geomview the plot y vs. t vs. 0. Again, copy this resulting target. Finally, send to Geomview the plot z vs. t vs. 0. The resulting plot in Geomview will be the overlay of the three plots. If you fail to copy any of the targets, they will be erased. This is because DsTool sends trajectory objects to Geomview using the name "Traj.n.m", where $n$ is the flow number, and $m$ is the trajectory number. Since we're only using one trajectory to generate all three plots, DsTool sends the data to the same target name each time, writing over the previous data in that target.

### 3.2.2 Tips for the Tcl/Tk Interface in DsTool

DsTool's interface is written in Tcl/Tk. In addition to providing an easy-to-use graphics interface framework, Tcl/Tk is a scripting language, and you can run DsTool entirely by entering commands at the prompt `%DsTool`. Some other useful tips follow:

1. Enter the command `end_wait` if DsTool grabs the input and does not relinquish it. Typically, during computations, DsTool disables mouse input and returns control to the mouse when the computation is finished. If an unexpected error occurs in the computation, control might not be properly restored. The command `end_wait` typed at the DsTool command prompt should return control to the mouse.

2. You can modify the default window colors and fonts in DsTool by following the instructions for adding custom Tcl/Tk code to your own version of DsTool. See the accompanying documentation in `$DSTOOL/my_dstool/README` for more information. The procedure essentially involves adding one line of code, and compiling your own version of DsTool.

3. While you can run DsTool entirely from the keyboard, it is not recommended that you do so. Typically, a button press will call "tcl_to_pm" to update the Postmaster, then "pm EXEC" to perform the computation by calling a C routine, and finally "pm_to_tcl" to update the interface. However, it may perform other tasks as well. For example, the "Forwards" button on the Continuation window checks to see if the number of active parameters selected is the correct number before proceeding. If you insist on running DsTool by typing from the prompt, you should use the procedures that Tcl/Tk calls when a button is pressed, instead of calling the Postmaster routines directly. For example, to replicate a press of the "Forwards" button on the Orbits window, instead of typing `pm EXEC Flow.Forwards`, type `orbits(forwards)`. In addition, if you change an entry in a field, you should follow it with a call of "tcl_to_pm" to update the Postmaster. Similarly, if you change a Postmaster entry, you should follow it with a call of "pm_to_tcl", to update the interface.

4. One efficient way to automate long calculations is by writing Tcl scripts. You can run scripts by entering the command "source *filename*" at the prompt. As with the previous note, you should be careful when writing scripts to make sure that you are computing the quantities that you wish to compute.

## 3.3 Colors in DsTool

### 3.3.1 Color Modes and Viewing

DsTool has two color modes: alternating color and picked color. The alternating color cycles through a list of entries in a colormap. For example, different colors are used to distinguish flows. The pick color can be selected by the user and is used to color code points according to the type of phenomena they represent.

Suppose the colormap has $N$ entries. In the default settings for DsTool, we have the bounds $4 \leq N \leq 13$. The value 4 is fixed, as DsTool needs at least 3 "system" colors, and at least one color for alternating colors. The value 13 is variable, and depends upon the value of the constant `MAX_DSCOLOR` in the file `$DSTOOL/src/include/dscolor.h`. Entries 0, 1, 2 of the colormap are reserved for system colors Red, Green, Blue, respectively. The alternating color cycles through entries 3 through $N-1$, while pick color can use any entry 0 through $N-1$. The available colors are displayed and can the pick color be selected in the Color window.

Each point that is stored in memory has three color (integer) entries. They are used to designate how the point is displayed in 2-D View windows. Points sent to Geomview via the 3-D View window are always displayed with dots, and points displayed in 1-D View windows are always connected with lines. The first color field contains the alternating color for the point, the second color field contains the pick color, and the third color field contains the symbol code. If the alternating color is a positive number, and the pick color setting on the 2-D or 3-D View window is off, then the point is plotted with the corresponding alternating color of the colormap. If the alternating color is negative, or the pick color setting of the 2-D or 3-D View window is on, then the point is plotted with the pick color. Regardless of the pick color setting, the point is plotted in 2-D View windows with the symbol determined by the symbol code.

## 3.3.2 Color Coding for Points

This section details how points are assigned colors and symbols.

**Colors for Trajectories:** Trajectories (and multiple trajectories) have both an alternating color and a pick color. The pick color can be chosen by the user using the Color window. The alternating color is automatically the next (non-system) color in the colormap. Each trajectory of a flow object is colored using the same alternating and pick colors. The default symbol (shown in the Defaults window) is always assigned.

**Colors for fixed points:** The DsTool convention for hyperbolic fixed points is: saddle points are marked as green crosses, sinks are blue triangles, and sources are red squares. In each case, the alternating color is negative, and the pick color and symbol code are the appropriate integers. Hyperbolic fixed points that are found by the Fixed Points window always follow the convention. Hyperbolic equilibrium points that are found by the Equilibrium Continuation window may or may not be colored according to the convention, depending upon the value of the Monitor Switch setting. In either setting, the equilibrium points are assigned dots, save those found by the "Search" operation, which are assigned triangles.

**Colors for periodic orbits:** Periodic orbits are colored using the DsTool convention: attracting periodic orbits are colored blue, repelling periodic orbits are colored red, and saddle periodic orbits are colored green. Periodic orbits are always assigned dots.

**Colors for bifurcation points:** Bifurcation points found by the Equilibrium Continuation window will be colored according to the colors in the Continuation Colors window. Saddle node bifurcation points are colored green. Hopf bifurcation points, degenerate Hopf points, and resonant saddle node points are colored using the last three colors in the colormap, which can be changed by the user. In the default colormap, the corresponding colors are magenta, orange, and sea green, respectively. Bifurcation points are always assigned dots, save those found by the "Search" operation, which are assigned triangles.

**Colors for selected points:** Points which are saved from the Selected Point window are assigned the pick color designated by the Color window, negative alternating color, and the default symbol.

## 3.3.3 Setting Up a User Defined Colormap

DsTool has a default colormap which it will use if the user does not provide one. To provide a colormap, the user must define it in a file named "rgb_color.txt". DsTool will check for this file first in the directory specified by the UNIX environmental variable `MY_DSTOOL`. If this fails, DsTool tries in the `DSTOOL` directory, and finally in the current working directory. If the file is not found in any of these places, the default colormap will be used. An example colormap looks like:

```
6
```

```
255   0   0
0 255   0
0   0 255
238 130 238
255 165   0
190 190 190
```

The first line gives the number of entries. This number must be correct. The next lines give the colors of the colormap, by specifying for each color an RGB triple. That is, each line consists of three integers between 0 and 255, which give the red, green, and blue components of the color. The file must not have any extraneous information such as comments, though it can have extra white space.

It is recommended that you use a colormap with at least six entries in it. This will ensure that all bifurcation phenomena that DsTool can detect will use colors different from the system colors. A colormap with less than four entries will not load, and DsTool will use the default colormap.

# Chapter 4

# User-Defined Dynamical Systems

In most instances, we expect that DsTool will be installed on a network of computers and used by a variety of researchers. It is not required, however, that each user on a network utilize the same executable version of DsTool. We have provided ways in which each user may customize certain features of DsTool to suit his or her needs. An example of DsTool customization is the installation of a user-defined dynamical system. In this chapter we will detail this process. More advanced customization, such as adding computational code to DsTool or adding animation to models, is discussed in `$DSTOOL/my_dstool/README`.

## 4.1 Preliminaries

We presume DsTool is installed according to the instructions provided in the DsTool installation information. This procedure includes:

1. Defining UNIX environmental variables `DSTOOL`, `MY_DSTOOL`, and `ARCH`.
2. Creating a local directory for DsTool.
3. Copying the contents of `$DSTOOL/my_dstool` from the DsTool source code into the above subdirectory.

Be certain that these steps are complete before proceeding.

## 4.2 Installing a New Dynamical System in DsTool

The addition of a new dynamical system in DsTool is a two-step process. The first step entails writing a few procedures which define the set of governing equations for the dynamical system (be it a vector field or a mapping) and the initial settings of variables and parameters. If desired, additional procedures may be written which define derivatives (with respect to space, time, and parameters) and define an arbitrary number of auxiliary scalar-valued functions. The second step in the process is to install the procedures into the libraries used to construct the executable version of DsTool. To help you complete the necessary steps, we provide the following checklist:

1. Define the dynamical system
   - (a) Equations of motion
   - (b) Derivatives
   - (c) Inverse
   - (d) Auxillary equations

(e) Names and default ranges for variables

(f) Names and default ranges for parameters

(g) Names and default ranges for auxiliary functions

(h) Periodic variables

(i) Names of user-defined functions

2. Install the dynamical system

(a) Initialization routine

(b) Title

(c) Compile source code

The discussion which follows contains details of this process and a description of the files and variables involved. If you are not already there, change directories to your local DsTool directory.

### 4.2.1 Defining the Equations of Motion

We shall use as an example a two-dimensional mapping $f = f_{\alpha,\gamma}$ defined by

$$\begin{aligned} \phi_{j+1} &= \phi_j + v_j, \\ v_{j+1} &= \alpha v_j - \gamma \cos(\phi_j + v_j). \end{aligned} \tag{4.1}$$

These equations are a model for the dynamics of a ball bouncing on a sinusoidally vibrating table. The variable $\phi$ is proportional to the time, but because of the sinusoidal motion of the table, we may think of $\phi$ as belonging to the circle parametrized by $[0, 2\pi)$. The variable $v$ is proportional to the ball's velocity immediately after contact with the table. The parameter $\alpha$, $0 < \alpha \leq 1$ is the coefficient of restitution for the ball; the parameter $\gamma > 0$ is the amplitude of the table oscillations. Essentially, the equations say that given the time of the $j$th impact ($\phi_j$) and the ball's velocity at that time ($v_j$), we know the time that the ball will next strike the table ($\phi_{j+1}$), as well as the ball's velocity immediately after its next bounce ($v_{j+1}$). See [1] for information about the dynamics of this system.

Suppose that a user wishes to study Equation 4.1 numerically using DsTool. The first task is to define the equations. First, copy the file `GENERIC.c` to the file that you want to edit. For this example, call the target file `bball_def.c`. You can use any text editor to edit `bball_def.c`. The beginning of the file looks like:

```
#include <model_headers.h>

/* ------------------------------------------------------------------------
   function used to define the vector field or map
   ------------------------------------------------------------------------ */
int user_ds_func(f,x,p)
double *f,*x,*p;
{
}
```

First change the name of this function from user_ds_func() to bball(). Then edit the function so that it properly defines the mapping. When you are finished, the function should look like:

```
#include <model_headers.h>

/* ------------------------------------------------------------------------
   function used to define the map
   ------------------------------------------------------------------------ */
int bball(f,x,p)
```

```
double *f,*x,*p;
{
   f[0] = x[0] + x[1];
   f[1] = p[0] * x[1] - p[1] * cos(x[0] + x[1]);
}
```

The mapping is now defined. We remind novice C-programmers that arrays in C are indexed from 0. When bball() is called, the calling routine is expected to pass in arrays `f`, `x`, and `p`. The input variables are the current state of variables (`x`) and the current parameters (`p`):

$$\begin{aligned} \mathtt{x} &= \{\mathtt{x[0]},\mathtt{x[1]}\} = \{\phi_j, v_j\}, \\ \mathtt{p} &= \{\mathtt{p[0]},\mathtt{p[1]}\} = \{\alpha, \gamma\}. \end{aligned}$$

The function bball() places the new state in the array `f`:

$$\mathtt{f} = \{\mathtt{f[0]},\mathtt{f[1]}\} = \{\phi_{j+1}, v_{j+1}\}.$$

This function *must* be defined in order to begin exploration of dynamics. If the user does not wish to input information about the system's derivative or inverse, and if no auxiliary functions are desired, the user could proceed to Section 4.2.5. For the purpose of illustration, however, we encourage the reader to continue reading.

We remark at this point that although this system is a two-dimensional mapping, the value $x[2]$ contains the current "time." In general, if there are $k$ dependent variables, then `x[0]`, ..., `x[k-1]` contain these variables and `x[k]` contains the independent variable. This is true both for maps and for vector fields.

### 4.2.2 Defining Derivative Information

If available, DsTool can use analytic information in certain computational routines. Jacobian information is used in root-finding algorithms and for implicitly iterating diffeomorphisms backwards (when an explicit inverse does not exist). Detailed information about these algorithms is found in the DsTool Reference Manual.

If derivative information is not provided by the user, then DsTool will use finite difference-approximations. This section will describe how to write a function which provides an explicit Jacobian for Equation 4.1. Because time is discrete for maps, it does not make sense to define a derivative with respect to time. Currently, DsTool does not make use of derivatives with respect to time and parameters, but we include them for the convenience of the user who wishes to extend the capabilities of DsTool.

We now continue with the bouncing ball example by defining the Jacobian of $f$. At the $j$th instant of time, the Jacobian is

$$\begin{pmatrix} \partial f_1/\partial \phi_j & \partial f_1/\partial v_j \\ \partial f_2/\partial \phi_j & \partial f_2/\partial v_j \end{pmatrix} = \begin{pmatrix} 1 & 1 \\ \gamma \sin(\phi_j + v_j) & \alpha + \gamma \sin(\phi_j + v_j) \end{pmatrix}$$

Find the section of the file `bball_def.c` which reads

```
/* ------------------------------------------------------------------------
   function used to define the Jacobian
   ------------------------------------------------------------------------ */
/*
int user_jac(m,x,p)
double  **m, *x, *p;
{
}
*/
```

Using a text editor, modify this code segment to read

```
/* ----------------------------------------------------------------------
   function used to define the Jacobian
   ---------------------------------------------------------------------- */
int bball_jac(m,x,p)
double  **m, *x, *p;
{
   double    temp;

   temp    = p[1] * sin( x[0] + x[1] );
   m[0][0] = 1.0;
   m[0][1] = 1.0;
   m[1][0] = temp;
   m[1][1] = p[0] + temp;
}
```

The routine which calls bball_jac() sends in a matrix and two arrays which contain the current state and the current parameters for $f_{\alpha,\gamma}$. When bball_jac() returns, it has filled the matrix with the numerical Jacobian for $Df_{\alpha,\gamma}$ evaluated at the current state. As remarked previously, writing a Jacobian routine is optional.

### 4.2.3 Defining Information About an Inverse

In this section we show how to define an inverse or an approximate inverse for a mapping. We recall that the "inverse function" for a vector field is trivial, since integrating the equations of motion backwards in time is the same as integrating forward in time—except we pass in a negative time step to a numerical integrator. Thus if we were installing a vector field, we would skip this section. Moreover, it frequently happens that a mapping does not have an inverse, nor is there any sort of an approximate inverse. In that case, do not edit the inverse routine at all.

For diffeomorphisms, it is sometimes difficult to iterate backwards in time. The Reference Manual discusses Newton's method and implicitly iterating a mapping backwards; it also briefly indicates some of the ways that Newton's method can fail. One of the most important aspects of Newton's algorithm is its dependence on a "good" initial guess. In the present section, we will discuss one way to provide a good guess.

For our bouncing ball example, it turns out that we can find an explicit inverse for $f$ since $\alpha > 0$. This inverse is the map given by

$$\begin{aligned} \phi_{j-1} &= \phi_j - \tfrac{1}{\alpha}(\gamma \cos \phi_j + v_j), \\ v_{j-1} &= \tfrac{1}{\alpha}(\gamma \cos \phi_j + v_j). \end{aligned} \tag{4.2}$$

To enter this inverse into DsTool, locate the section of `bball_def.c` which looks like

```
/* ----------------------------------------------------------------------
   function used to define inverse or approximate inverse
   ---------------------------------------------------------------------- */
/*
int user_inv(y,x,p)
double    *y,*x,*p;
{
}
*/
```

and modify this section to produce

```
/* ----------------------------------------------------------------------
   function used to define inverse
   ------------------------------------------------------------------------ */
int bball_inv(y,x,p)
double    *y,*x,*p;
{
   double     temp;

   temp = ( p[1] * cos( x[0] ) + x[1] ) / p[0];
   y[0] = x[0] - temp;
   y[1] = temp;
}
```

Suppose that our map did not have an explicit inverse (or we could not calculate it). Then we would have to resort to using Newton's method to numerically solve for the inverse iterate at each backward step. To do so requires that we provide an initial guess for the inverse image of the current state. If we provide a good guess for the inverse image, then possibly we will require only a few Newton steps to "sharpen" our guess and converge to a solution (See the Reference Manual). Suppose, for the sake of an example, that we were only interested in the bouncing ball map $f_{\alpha,\gamma}$ in the special case that $\gamma$ is small. Then $f$ can be thought of as a perturbation of the linear map $g$ defined by

$$\begin{aligned} \phi_{j+1} &= \phi_j + v_j, \\ v_{j+1} &= \alpha v_j, \end{aligned} \tag{4.3}$$

and $g^{-1}$ can be written as:

$$\begin{aligned} \phi_{j-1} &= \phi_j - \tfrac{1}{\alpha} v_j, \\ v_{j-1} &= \tfrac{1}{\alpha} v_j. \end{aligned} \tag{4.4}$$

In this case, we could code $g^{-1}$ into DsTool in place of $f^{-1}$. If we were to do this, we would modify `bball_def.c` to read:

```
/* ------------------------------------------------------------------------
   function used to define approximate inverse
   ------------------------------------------------------------------------ */
int bball_approx_inv(y,x,p)
double    *y,*x,*p;
{
   y[0] = x[0] - x[1] / p[0];
   y[1] = x[1] / p[0];
}
```

Of course, we must inform DsTool whether to interpret the procedure as a definition of the actual inverse or merely as an approximation; Section 4.2.5 explains how this is done.

### 4.2.4 Defining Auxiliary Functions

We continue with the above example. Suppose that there is some scalar function of the phase space variables, time, and parameters which we wish to monitor. The scalar function may be a physically meaningful quantity such as the total energy of an (almost) conservative system, or it could be a mathematically interesting quantity such as the largest eigenvalue of the Jacobian matrix. Sometimes it is useful to have functions which represent coordinate transformations. For example, a particular dynamical system may be best described in terms of spherical coordinates. Auxiliary functions can also be used to monitor the distance from some set of interest. For example, if a system spends most of its time close to the sphere $|x| = 1$ then it may be interesting to define a function $|x|$.

For our bouncing ball example, we will monitor the quantity $v^2$. Since $v_j$ is proportional to the ball's velocity just after it strikes the table for the $j$th time, we can think of $v_j^2$ as being a measure of the ball's kinetic energy at time $j$. To define this function, find the location in the file `bball_def.c` which reads

```
/* ----------------------------------------------------------------------
   function used to define aux functions of the varbs, time, or params
   ---------------------------------------------------------------------- */
/*
int user_aux_func(f,x,p)
double *f,*x,*p;
{
}
*/
```

and edit it to produce

```
/* ----------------------------------------------------------------------
   function used to define aux functions of the varbs, time, or params
   ---------------------------------------------------------------------- */
int bball_aux(f,x,p)
double *f,*x,*p;
{
   f[0] = x[1] * x[1];
}
```

The number of auxiliary functions is completely arbitrary; it is not necessary to have any auxiliary functions. When this equation is called, the calling routine passes in the current state of the phase space variables (in `x`), the current parameters (in `p`), and an array (`f`) which is filled up by this function.

## 4.2.5 Defining Labels and Initial Conditions

This section explains how to complete the definition of a dynamical system by giving DsTool information about the structure of phase space, the names of variables, and initial conditions.

This section is split into subsections. The subsections describe how to provide information on variables, parameters, auxiliary functions, periodic variables, and numerical algorithms.

Before we begin, find the code segment of `bball_def.c` which looks like

```
/* ----------------------------------------------------------------------
   Procedure to define default data for the dynamical system. NOTE: You
   may change the entries for each variable but DO NOT change the list of
   items.  If a variable is unused, NULL or zero the entry, as appropriate.
   ---------------------------------------------------------------------- */
int user_init()
{
/* ----------- define the dynamical system in this segment -------------- */
...
```

and change the name of the function from user_init() to bball_init().

### 4.2.5.1 Variables

For our example, we have two spatial phase space variables, namely $\phi$ and $v$. Because the equations of motion (Equation 4.1) are invariant under the coordinate transformation $\phi \to \phi + 2n\pi$, $n \in \mathbb{Z}$, it is natural to display $\phi$ on the interval $[0, 2\pi]$. There is no "natural" interval to use in displaying the $v$ coordinate, since $v$ can be any real number, but we will choose the interval $[-30, 30]$ as a default range on which to display $v$. We will also need to choose a default initial condition $(\phi_0, v_0)$, which we arbitrarily select to be the origin, $(0, 0)$.

After implementing these choices, the relevant code in bball_init() looks like:

```
int            n_varb=2;                    /* dim of phase space          */
static char    *variable_names[]={"phi","v"}; /* list of phase varb names    */
static double  variables[]={0.,0.};          /* default varb initial values  */
static double  variable_min[]={0.,-30.};     /* default varb min for display */
static double  variable_max[]={TWOPI,30};    /* default varb max for display */
```

We remark that TWOPI ($2\pi$) and PI ($\pi$) are two constants which the user may use in defining a dynamical system.

Although we have defined labels and initial values for the *spatial* variables, the independent variable (usually thought of as time) is also considered to be a member of every phase space. The code which provides this information is given by:

```
static char    *indep_varb_name="time";  /* name of indep variable             */
static double  indep_varb_min=0.;        /* default indep varb min for display */
static double  indep_varb_max=10000.;    /* default indep varb max for display */
```

In fact, this is the way the code looked when we copied it from GENERIC.c, so we do not need to make any changes to the code. If we wanted to call the independent variable "iteration" instead of "time," or if we wanted to change the default plotting range, then the code segment above would have to be appropriately modified.

#### 4.2.5.2 Parameters

When examining bifurcations of our bouncing ball map, it is useful to graphically depict (a subset of) the parameter space. Since $0 < \alpha \leq 1$, we choose the interval $[0, 1]$ as the default range to display $\alpha$. For $\gamma$, we choose $[0, 25]$ as a default range. As initial parameter values, we choose $(\alpha, \gamma) = (0.8, 10)$, since the dynamics for these parameter values are fairly interesting.

To implement these choices, we edit a few more lines in bball_init() so that the code looks like:

```
int            n_param=2;               /* dim of parameter space         */
static char    *parameter_names[]={"alpha","gamma"}; /* list of param names */
static double  parameters[]={0.8,10.};   /* initial parameter values       */
static double  parameter_min[]={0.,0.};  /* default param min for display   */
static double  parameter_max[]={1.,25.}; /* default param max for display   */
```

#### 4.2.5.3 Auxiliary Functions

For our example, we have only a single function which we will call "KE" for kinetic energy. From the definition of the function ($v^2$) and from the default plotting range for the variable $v$, the interval $[0, 1000]$ is probably a suitable range on which to plot the value of $v^2$. Accordingly, we edit a few more lines in bball_init():

```
int            n_funct=1;               /* number of user-defined functions */
static char    *funct_names[]={"KE"};    /* list of funct names; {""} if none*/
static double  funct_min[]={0};         /* default funct min for display    */
static double  funct_max[]={1000};      /* default funct max for display    */
```

We remark that if we did not want to monitor any auxiliary functions then we would set n_funct=0. The array of function names, however, must contain at least an empty string or else our code will not compile properly. In other words, if there were no auxiliary quantities of interest, then we could write *funct_names[]="", but *not* *funct_names[]=.

#### 4.2.5.4 Periodic Variables

We must tell DsTool whether our phase space contains any periodic variables. The coding of this information is accomplished through two C-language constants: EUCLIDEAN or PERIODIC. If there are no periodic variables, then the phase space is EUCLIDEAN; if there is at least one periodic variable, the phase space is classified as PERIODIC. For PERIODIC phase spaces, we must also inform DsTool as to which variables are periodic and what interval to use as a fundamental domain.

For our example, since the equations of motion are invariant under the transformation $\phi \to \phi + 2n\pi$ for any integer $n$, we may consider $\phi$ to be a periodic variable with period $2\pi$. We may choose *any* interval of length $2\pi$ as a fundamental domain for the variable $\phi$. Common choices are the intervals $[-\pi, \pi]$ and $[0, 2\pi]$. We make the latter choice. To pass this information into DsTool, we edit yet a few more lines in bball_init():

```
int           manifold_type=PERIODIC;   /* EUCLIDEAN or PERIODIC               */
static int    periodic_varb[]={TRUE, FALSE}; /* if PERIODIC, which varbs periodic? */
static double period_start[]={0.,0.};   /*if PERIODIC, begin fundamental domain */
static double period_end[]={TWOPI, 1.}; /*if PERIODIC, end of fundamental domain*/
```

We remark on the variables period_start and period_end. If the $j$th coordinate is not periodic (*i.e.*, the value of periodic_varb[$j$] is FALSE) then it does not matter what period_start[$j$] and period_end[$j$] are because the entries are ignored by DsTool. Similarly, if the variable manifold_type is EUCLIDEAN, then it doesn't matter what values are given for the entries of periodic_varb. It is always safe, of course, to set each entry of periodic_varb to FALSE. As mentioned in Section 4.2.5.1, TWOPI is a global constant.

#### 4.2.5.5 Numerical Algorithms

Continuing with our example, there remain a few pieces of information which we have not yet entered into DsTool. We modify the last lines of code in bball_init() to obtain:

```
int        mapping_toggle=TRUE;         /* this is a map? TRUE or FALSE        */
int        inverse_toggle=EXPLICIT_INV; /* if so, is inverse FALSE, APPROX_INV, */
                                        /* or EXPLICIT_INV? FALSE for vec field */

/*  In this section, input NULL or the name of the function which contains...   */
int     (*def_name)()=bball;            /* the eqns of motion                   */
int     (*jac_name)()=bball_jac;        /* the jacobian (deriv w.r.t. space)    */
int     (*aux_func_name)()=bball_aux;   /* the auxiliary functions              */
int     (*inv_name)()=bball_inv;        /* the inverse or approx inverse        */
int     (*dfdt_name)()=NULL;            /* the deriv w.r.t time                 */
int     (*dfdparam_name)()=NULL;        /* the derivs w.r.t. parameters         */
```

The first line means that our system is a mapping; if it were a vector field, we would set mapping_toggle to FALSE. The second line means that we have an explicit inverse for our system. In the event that a mapping does not have an explicit inverse, then Newton's method will be used to calculate inverse iterates. In this event, we need to tell DsTool if there exists an approximate inverse (see Section 4.2.3 or the Reference Manual) which can be used as an initial guess for Newton's method. If there is, then inverse_toggle is set to APPROX_INV; otherwise, it is set to FALSE. We remark that if mapping_toggle is set to FALSE (that is, the system is a vector field), then the value of inverse_toggle is ignored by DsTool.

In the last section of code, we provide DsTool with the names of any functions we defined previously. For our example, this means filling in the names of the functions which compute the equations of motion, the Jacobian, auxiliary functions, and the inverse. If any of these functions were omitted, then the user should enter NULL instead of a name. For example, the function which defines the inverse will be NULL if you are installing a vector field, and the function which defines the derivative with respect to time should always be NULL for mappings.

At this point, the user should save and exit the file bball_def.c.

### 4.2.6 Installing a Defined Dynamical System

It is assumed that the reader has completed the steps in the preceding sections of this chapter. In this section, we describe the installation of the dynamical system defined using the above steps. As an illustration, we continue with the bouncing ball example.

Our first task is to modify the file `user.c`. There are two steps to this process:

- Inform DsTool about the name of the initialization routine for the dynamical system we desire to install. In our bouncing ball example, this function is bball_init().
- Enter this name into the DsTool list of dynamical systems, along with a title.

During the execution of DsTool, the user may select one of several installed dynamical systems. The program maintains a list of installed models, along with the procedures which initialize and define them. The user-defined models are maintained in the file `user.c` of the user's local DsTool directory. This file contains three relevant blocks of C code: the first block defines an array of categories used to collect dynamical systems into related classes, the second consists of several lines beginning with the declaration `extern int`, the last block is a C-language structure which contains titles for the models along with names of the models' initialization procedures. The relevant pieces of the file `user.c` delivered with DsTool contain the following code:

```
/* ----------------------------------------------------------------
 * INCLUDE USER DYNAMICAL SYSTEMS CATEGORIES HERE
 *
 * ----------------------------------------------------------------
 */
char *USER_DS_Category[] = { "User models" /* Category 0 */
                           };

/* ----------------------------------------------------------------
 * INCLUDE USER DYNAMICAL SYSTEM NAMES HERE
 *
 * We have put in the Lorenz system as an example
 * ----------------------------------------------------------------
 */

/* declare the model initialization procedure here */
extern int lorenz_init();


/* list the category, a short model name, and the initialization proc here */
struct DS_DataS USER_DS_Sel[]= {
  { 0, "Lorenz system", lorenz_init }
};
```

To tell DsTool the name of the initialization routine which defines our bouncing ball model, we need to add the line

```
extern int  bball_init();
```

to the second block of code in `user.c`. To install our model, we replace the line of code

```
  { 0, "Lorenz system", lorenz_init }
```

with the line of code

```
  { 0, "Bouncing Ball", bball_init }
```

in the definition of the data structure `USER_DS_Sel` within the third block of code. `USER_DS_Sel` is an array, each element of which is a data structure which defines a dynamical system. The first element of the structure is a number which specifies the user category to which the dynamical system belongs. The category is used to group together systems which share similar properties. These categories are defined in the first block of code. Users cannot add dynamical systems to the standard list of categories. However, they can create as many new categories as they wish, by modifying the category list like so:

```
char *USER_DS_Category[] = { "User models", /* Category 0 */
                             "Project with Bob", /* Category 1 */
                             "My favorite dynamical systems" /* Category 2 */
                           };
```

This array provides the title for each category. For our example, the category index is 0, so our bouncing ball model belongs to the category named "User models." (Recall that arrays in C are indexed from 0 so `DS_Category`[0] is not an invalid array index.) Every dynamical system must belong to a valid category.

The next element of the dynamical system data structure is the title of the system being defined. We have chosen "Bouncing Ball" as the title of our system. Any descriptive title will do, provided that the length of the title is not larger than the global constant `MAX_LEN_DS_TITLE`. The value of this constant is found in the file `$DSTOOL/src/include/constants.h`.

The last element is the name of the function which contains the initialization routine for the new dynamical system. For our example, the name is `bball_init`.

Note that each dynamical system data structure must be enclosed by braces and followed by a comma. That is, all except the last. The last element in the array of dynamical systems (our new dynamical system "Bouncing Ball" in our example code) should not have a trailing comma.

We are now ready to compile the source code we have written. To do this, edit the `Makefile` in your local DsTool directory. Add the file `bball_def.c` to the `USER_SRCS` list of files, and the file `bball_def.o` to the `USER_OBJS` list of files. In the example described above, the corresponding Makefile entry would be:

```
USER_SRCS = user.c bball_def.c
USER_OBJS = user.o bball_def.o
```

Now save and exit the `Makefile`.

If our model compiles without errors, we can create a custom version of DsTool called `my_dstool`. To do this, execute the command `make` in your local DsTool directory.

Once this is done, you can ensure that this version of DsTool is executed by settting the UNIX environmental variable `MY_DSTOOL` to your local DsTool directory. When the script `dstool_tk` is run from any directory, it will execute the binary `$MY_DSTOOL/bin/$ARCH/my_dstool`. A message will be printed confirming which binary is being executed. This version of DsTool will include the bouncing ball equations among its installed dynamical systems.

## 4.3 A Vector Field Example: Duffing's Equations

In this section we present an example of installing a user-defined vector field into DsTool. It is assumed that the reader is familiar with the material in Section 4.2.

Our task is to install the vector field defined by the equations

$$\begin{aligned} \dot{u} &= v, \\ \dot{v} &= u - u^3 - \delta v - \gamma \cos \omega t. \end{aligned} \tag{4.5}$$

This system is known as Duffing's equations and we direct the reader to [1] and references therein for an exposition of the dynamics of this system.

To define and install this system, change to your local DsTool directory and copy the file `GENERIC.c` to the file `userduffing_def.c`. Use any text editor to edit `userduffing_def.c`.

The segment of code which defines the equations of motion (Equation 4.5) will be called userduffing(). This function is defined as follows:

```
/* ------------------------------------------------------------------------
   function used to define the vector field
   ------------------------------------------------------------------------ */
int userduffing(f,x,p)
double *f,*x,*p;
{
   f[0] = x[1];
   f[1] = x[0] - x[0]*x[0]*x[0] - p[0]*x[1] - p[1]*cos( p[2]*x[2] );
}
```

Here we have defined the variables and parameters as

$$\begin{aligned} \mathtt{x} &= \{\mathrm{x}[0], \mathrm{x}[1]\} = \{u, v\}, \\ \mathtt{p} &= \{\mathrm{p}[0], \mathrm{p}[1], \mathrm{p}[2]\} = \{\delta, \gamma, \omega\}. \end{aligned}$$

As usual, the independent variable is stored after the spatial variables so that `x[2]` is time. The function userduffing() returns in the array `f` the strength of the vector field with parameters `p`, evaluated at the point `x`.

We now define the Jacobian of Equation 4.5 by writing the function userduffing_jac():

```
/* ------------------------------------------------------------------------
   function used to define the Jacobian
   ------------------------------------------------------------------------ */
int userduffing_jac(m,x,p)
double  **m, *x, *p;
{
   m[0][0] = 0.0;
   m[0][1] = 1.0;
   m[1][0] = 1.0 - 3.0 * x[0] * x[0];
   m[1][1] = -p[0];
}
```

Since our equations are for a vector field, we do not need to define an inverse function. Our vector field is time-dependent, so we can define a function which returns the temporal derivatives of the vector field. This function is not yet used by DsTool, and there is no template for such a function in `GENERIC.c`, but it is easy enough to write:

```
/* ------------------------------------------------------------------------
   function used to define temporal derivatives
   ------------------------------------------------------------------------ */
int userduffing_dfdt(f,x,p)
double  *f, *x, *p;
{
   f[0] = 0.0;
   f[1] = -p[1] * p[2] * sin( p[2] * x[2] );
}
```

Since our vector field is Hamiltonian for $\delta = 0$, we choose

$$H(u, v) = \frac{v^2}{2} - \frac{u^2}{2} + \frac{u^4}{4}$$

as an auxiliary function. We will also define a second auxiliary function. Since our vector field is periodic in time (with period $2\pi/\omega$), it is a common technique to look at the time-$2\pi/\omega$ map in order to better understand the dynamics. This "stroboscopic map" can also be thought of as a Poincaré map for the extended phase space. We are thus interested in the times $t$ for which $\sin(wt + t_0) = 0$. Choosing $t_0 = 0$ we define the procedure userduffing_aux() by:

```
/* ----------------------------------------------------------------------
   function used to define aux functions of the varbs, time, or params
   ---------------------------------------------------------------------- */
int userduffing_aux(f,x,p)
double *f,*x,*p;
{
   double   temp;

   temp = 0.5 * x[0] * x[0];
   f[0] = 0.5 * x[1] * x[1] - temp + temp * temp;
   f[1] = sin( p[2] * x[2] );
}
```

We are now ready to define the labels and initial conditions for Duffing's equations by writing the function userduffing_init(). We choose $[-2, 2]$ as default plotting ranges for both $u$ and $v$. Initializing the variables is straightforward:

```
int userduffing_init()
{
/* ------------ define the dynamical system in this segment --------------- */

int            n_varb=2;                  /* dim of phase space           */
static char    *variable_names[]={"u","v"}; /* list of phase varb names   */
static double  variables[]={0.5,0.5};     /* default varb initial values  */
static double  variable_min[]={-2.,-2.};  /* default varb min for display */
static double  variable_max[]={2.,2.};    /* default varb max for display */

static char    *indep_varb_name="time";   /* name of indep variable             */
static double  indep_varb_min=0.;         /* default indep varb min for display */
static double  indep_varb_max=1000;       /* default indep varb max for display */
```

Defining the parameter ranges is somewhat arbitrary. Sometimes it is difficult to tell *a priori* what range of parameters will provide interesting bifurcations. This is not a big problem since it is a trivial matter to change the range upon which a function (or parameter) is plotted once within DsTool. We choose $[0, 1]$ as a range for each parameter. We choose $(\delta_0, \gamma_0, \omega_0) = (0.25, 0.4, 1.0)$:

```
int            n_param=3;                 /* dim of parameter space       */
static char    *parameter_names[]={"delta","gamma", "w"}; /* list of param names */
static double parameters[]={0.25,0.4,1.}; /* initial parameter values     */
static double parameter_min[]={0.,0.,0.}; /* default param min for display */
static double parameter_max[]={1.,1.,1.}; /* default param max for display */
```

The initialization of our two auxiliary functions is accomplished by the code segment:

```
int            n_funct=2;               /* number of user-defined functions */
static char    *funct_names[]={"H", "P_Section"}; /* list of funct names; {""} if none */
static double  funct_min[]={-4.,-1.};   /* default funct min for display    */
static double  funct_max[]={4.,1.};     /* default funct max for display    */
```

As in the case with parameters, it is sometimes difficult to choose *a priori* what appropriate ranges for the functions should be.

The manifold type for Duffing's equations is `EUCLIDEAN` since we do not have any periodic spatial variables. Thus we do not need to modify the following segment of code:

```
int           manifold_type=EUCLIDEAN; /* EUCLIDEAN or PERIODIC                */
static int    periodic_varb[]={FALSE, FALSE}; /* if PERIODIC, which varbs periodic? */
static double period_start[]={0.,0.};  /* if PERIODIC, begin fundamental domain */
static double period_end[]={1., 1.};   /* if PERIODIC, end of fundamental domain*/
```

The last segment of code we need to modify is the segment which tells DsTool which numerical algorithms may be used on the Duffing's equations. We complete the definition of userduffing_init() with the code segment:

```
int           mapping_toggle=FALSE;       /* this is a map? TRUE or FALSE         */
int           inverse_toggle=FALSE;       /* if so, is inverse FALSE, APPROX_INV, */
                                        /* or EXPLICIT_INV? FALSE for vec field */

/*  In this section, input NULL or the name of the function which contains...   */
int     (*def_name)()=userduffing;          /* the eqns of motion                   */
int     (*jac_name)()=userduffing_jac;      /* the jacobian (deriv w.r.t. space)    */
int     (*aux_func_name)()=userduffing_aux; /* the auxiliary functions              */
int     (*inv_name)()=NULL;             /* the inverse or approx inverse        */
int     (*dfdt_name)()=userduffing_dfdt;    /* the deriv w.r.t time                 */
int     (*dfdparam_name)()=NULL;        /* the derivs w.r.t. parameters         */
```

As in Section 4.2, we now need to edit two other files. We edit `Makefile` and add `userduffing_def.c` to the `USER_SRCS` list, and add `userduffing_def.o` to the `USER_OBJS` list. We also edit the file `user.c` and add the lines

```
extern int  userduffing_init();
```

and

```
{0, "Duffing's Eqns", userduffing_init},
```

to the second and third blocks of code, respectively.

Typing `make` will create a local executable of DsTool which includes Duffing's equations among its installed dynamical systems.

### 4.3.1 A Few Remarks on the Definition of Duffing's Equations

Recall that when we installed Duffing's equations as a time-dependent vector field, we defined $\sin \omega t$ as an auxiliary function and claimed that we could use it to study the time-$2\pi/\omega$ stroboscopic map. In theory, there is nothing wrong with this, however in practice we will encounter numerical errors in the evaluation of transcendental functions such as $\sin \omega t$ for large values of $t$. Since we are often interested in generating Poincaré maps for extremely long times, and since the function $\cos \omega t$ also appears in the definition of our vector field, the user may want to extend phase space by introducing the variable $\theta$. Then we can rewrite Duffing's equations in the form

$$\begin{aligned} \dot{u} &= v, \\ \dot{v} &= u - u^3 - \delta v - \gamma \cos \omega\theta, \\ \dot{\theta} &= 1, \end{aligned} \tag{4.6}$$

where $(u, v, \theta) \in \mathbb{R}^2 \times S^1$, and $S^1$ is the circle of length $2\pi/\omega$. That is, $\theta$ takes values in $[0, 2\pi/\omega)$. The problem with this formulation is that DsTool *cannot handle periodic variables whose length depends on a parameter!* To overcome this difficulty, we change coordinates via the transformation $\psi = \omega\theta$. Thus we could study Duffing's equations on extended phase space in the form

$$\begin{aligned} \dot{u} &= v, \\ \dot{v} &= u - u^3 - \delta v - \gamma \cos \psi, \\ \dot{\psi} &= \omega, \end{aligned} \tag{4.7}$$

where $(u, v, \psi) \in \mathbb{R}^2 \times S^1$, and $S^1$ is now the circle of length $2\pi$.

The advantage to an extended phase space such as we have for Equation 4.7 is that it is trivial to plot Poincaré sections for this set of equations because we can make $\psi$ a periodic variable. This allows us to request that DsTool only plot points when $\theta = \theta_0$ for some $\theta_0$. In contrast, DsTool *never treats time as a periodic variable*, so we needed to define the auxiliary function $\sin \omega t$ in order to be able to generate a Poincaré map for Duffing's equations.

## 4.4 Deleting Dynamical Systems

Suppose that we want to remove the dynamical system "Bouncing Ball" from section 4.2, which is defined in the file `bball_def.c`. To remove this:

1. Edit the `Makefile` in your local DsTool directory. Delete the name of the file which contains the dynamical system from the `USER_SRCS` list and the name of the corresponding object file from the `USER_OBJS` list. In our example, we would remove the file names `bball_def.c` and `bball_def.o` from the `Makefile`.
2. Edit the file `user.c` and delete all references to the dynamical system you wish to remove. To remove the Bouncing ball system, delete the lines

   ```
   extern int bball_init();
   ```

   and

   ```
   {0, "Bouncing Ball", bball_init},
   ```

   from the file.
3. Remake your custom executable as explained in previous sections.

We note that you cannot delete any of the standard models that come with DsTool. You can only delete models that you have previously added to your own custom executable.

# Acknowledgements

We would like to thank Swan Kim for writing **kaos**, the precursor to DsTool, as well as the original version of the Parser window. Without his efforts, DsTool might never have been written. We have profited greatly from our experiences with kaos.

We thank Al Thaler and the computational mathematics section of the NSF for their support. Also the Army Research Office and the Mathematical Sciences Institute financed some of the equipment used for development.

We would also like to thank Allen Back and Toby Driscoll for their work on DsTool. Both Allen and Toby have worked on developing a three-dimensional view window; this work is in progress. Allen has also written the tutorial and has made numerous helpful suggestions.

Helpful comments and suggestions have also come from other students and visitors to the Center for Applied Mathematics. In particular we thank Moshe Braner, Feng-Zhao and Rob Manning for their useful ideas and enthusiasm.

The current version of DsTool could not have been written without the help of Nicolas Vera and Hinke Osinga at the Geometry Center.

# Appendix A

# Fonts

We present below a listing of fonts available using the Print window of DsTool.

| Font | Example Text |
|---|---|
| Times-Roman | 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<br>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 |
| *Times-Italic* | *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*<br>*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* |
| **Times-Bold** | **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**<br>**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** |
| ***Times-BoldItalic*** | ***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***<br>***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*** |
| Helvetica | 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<br>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 |
| *Helvetica-Oblique* | *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*<br>*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* |
| **Helvetica-Bold** | **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**<br>**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** |
| ***Helvetica-BoldOblique*** | ***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***<br>***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*** |
| ***Courier*** | ***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***<br>***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*** |
| *Courier-Oblique* | *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*<br>*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* |
| **Courier-Bold** | **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**<br>**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** |
| ***Courier-BoldOblique*** | ***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***<br>***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*** |

# Bibliography

[1] J. Guckenheimer and P. J. Holmes. *Nonlinear Oscillations, Dynamical Systems, and Bifurcations of Vector Fields.* Springer-Verlag, New York, 1983.

[2] J. Guckenheimer, M. Myers. "Computing Hopf Bifurcations II", SIAM J. Sci. Comp., 17, 1275-1301, 1996

[3] J. Guckenheimer, M. Myers, and B. Sturmfels. "Computing Hopf Bifurcations I", SIAM J. Num. Anal., 34, 1-21, 1997

[4] W. H. Press et al. *Numerical Recipes in C: The Art of Scientific Computing.* Cambridge UP, Cambridge, 1988.

[5] J. Stoer and R. Bulirsch. *Introduction to Numerical Analysis.* Springer-Verlag, New York, 1980.

# Index