

AUTO-07P :
CONTINUATION AND BIFURCATION SOFTWARE
FOR ORDINARY DIFFERENTIAL EQUATIONS

Eusebius J. Doedel
Concordia University
Montreal, Canada

with major contributions by

Alan R. Champneys (Bristol), Thomas Fairgrieve (Toronto),
Yuri Kuznetsov (Utrecht), Bart Oldeman, Randy Paffenroth (Pasadena),
Björn Sandstede (Surrey), Xianjun Wang, and Chenghai Zhang.

August 2007

Contents

1	Installing AUTO.	11
1.1	Installation.	11
1.1.1	Installation on Mac OS X	12
1.1.2	Installation on Windows	12
1.2	Restrictions on Problem Size.	13
1.3	Compatibility with Earlier Versions.	13
1.4	Parallel Version.	14
2	Overview of Capabilities.	15
2.1	Summary.	15
2.2	Algebraic Systems.	15
2.3	Ordinary Differential Equations.	16
2.4	Parabolic PDEs.	17
2.5	Discretization.	18
3	User-Supplied Files.	19
3.1	The Equations-File <code>xxx.f90</code> , or <code>xxx.f</code> , or <code>xxx.c</code>	19
3.2	The Constants-File <code>c.xxx</code>	19
3.3	User-Supplied Routines.	20
3.4	User-Supplied Derivatives.	20
4	Running AUTO using Python Commands.	21
4.1	Typographical Conventions	21
4.2	General Overview.	21
4.3	First Example	22
4.4	Scripting	26
4.5	Second Example	26
4.6	Extending the AUTO CLUI	30
4.7	Bifurcation Diagram Files	32
4.8	Solution Files	35
4.9	Exporting output data for use by Python or external visualization tools.	37
4.10	The <code>.autorc</code> File	37
4.11	Two Dimensional Plotting Tool	38
4.12	Three Dimensional Plotting Tool	40
4.13	Quick Reference	40

4.14	Reference	42
4.14.1	commandAppend	42
4.14.2	commandCat	42
4.14.3	commandCd	43
4.14.4	commandClean	43
4.14.5	commandCopyAndLoadDemo	44
4.14.6	commandCopyDataFiles	44
4.14.7	commandCopyDemo	45
4.14.8	commandCopyFortFiles	45
4.14.9	commandCreateGUI	46
4.14.10	commandDeleteDataFiles	46
4.14.11	commandDeleteFortFiles	47
4.14.12	commandDouble	47
4.14.13	commandInteractiveHelp	48
4.14.14	commandLs	48
4.14.15	commandMoveFiles	49
4.14.16	commandParseConstantsFile	49
4.14.17	commandParseDiagramAndSolutionFile	50
4.14.18	commandParseDiagramFile	50
4.14.19	commandParseSolutionFile	51
4.14.20	commandPlotter	51
4.14.21	commandPlotter3D	52
4.14.22	commandQueryBranchPoint	52
4.14.23	commandQueryEigenvalue	53
4.14.24	commandQueryFloquet	53
4.14.25	commandQueryHopf	54
4.14.26	commandQueryIterations	54
4.14.27	commandQueryLimitpoint	55
4.14.28	commandQueryNote	55
4.14.29	commandQuerySecondaryPeriod	56
4.14.30	commandQueryStepsize	56
4.14.31	commandRun	57
4.14.32	commandRunnerConfigFort2	58
4.14.33	commandRunnerConfigFort12	58
4.14.34	commandRunnerLoadName	59
4.14.35	commandRunnerPrintFort2	60
4.14.36	commandRunnerPrintFort12	60
4.14.37	commandShell	61
4.14.38	commandSpecialPointLabels	61
4.14.39	commandTriple	62
4.14.40	commandUserData	62
4.14.41	commandWait	63

5	Running AUTO using Unix Commands.	64
5.1	Basic commands.	64
5.2	Plotting commands.	65
5.3	File-manipulation.	65
5.4	Diagnostics.	65
5.5	File-editing.	66
5.6	File-maintenance.	66
5.7	HomCont commands.	67
5.8	Copying a demo.	67
5.9	Viewing the manual.	67
6	Output Files.	68
7	The Graphics Program PLAUT.	70
7.1	Basic PLAUT-Commands.	70
7.2	Default Options.	71
7.3	Other PLAUT-Commands.	72
7.4	Printing PLAUT Files.	72
8	The Graphics Program PLAUT04.	73
8.1	Quick start	73
8.1.1	Starting and stopping PLAUT04	73
8.1.2	Changing the “Type”	74
8.1.3	Changing the “Style”	74
8.1.4	Coordinate axes	74
8.1.5	Options	74
8.1.6	CR3BP animation	74
8.1.7	Help	75
8.1.8	Picking a point in the diagram	75
8.1.9	Choosing the variables	76
8.1.10	Choosing labels	78
8.1.11	Coloring	78
8.1.12	Number of periods to be animated	78
8.1.13	Changing the line/tube thickness	79
8.1.14	Changing the animation speed	79
8.1.15	Changing the background picture	79
8.2	Setting up the resource file	79
8.3	Example	82
9	The Graphical User Interface GUI94.	85
9.1	General Overview.	85
9.1.1	The Menu bar.	85
9.1.2	The Define-Constants-buttons.	85
9.1.3	The Load-Constants-buttons.	86
9.1.4	The Stop- and Exit-buttons.	86

9.2	The Menu Bar.	86
9.2.1	Equations-button.	86
9.2.2	Edit-button.	86
9.2.3	Write-button.	86
9.2.4	Define-button.	86
9.2.5	Run-button.	87
9.2.6	Save-button.	87
9.2.7	Append-button.	87
9.2.8	Plot-button.	87
9.2.9	Files-button.	87
9.2.10	Demos-button.	87
9.2.11	Misc.-button.	88
9.2.12	Help-button.	88
9.3	Using the GUI.	88
9.4	Customizing the GUI.	88
9.4.1	Print-button.	88
9.4.2	GUI colors.	88
9.4.3	On-line help.	89
10	Description of AUTO-Constants.	90
10.1	The AUTO-Constants File.	90
10.2	Problem Constants.	90
10.2.1	NDIM	90
10.2.2	NBC	90
10.2.3	NINT	91
10.2.4	JAC	91
10.3	Discretization Constants.	91
10.3.1	NTST	91
10.3.2	NCOL	91
10.3.3	IAD	91
10.4	Tolerances.	92
10.4.1	EPSL	92
10.4.2	EPSU	92
10.4.3	EPSS	92
10.4.4	ITMX	92
10.4.5	NWTN	92
10.4.6	ITNW	92
10.5	Continuation Step Size.	93
10.5.1	DS	93
10.5.2	DSMIN	93
10.5.3	DSMAX	93
10.5.4	IADS	93
10.5.5	NTHL	94
10.5.6	NTHU	94
10.6	Diagram Limits.	94

10.6.1	NMX	94
10.6.2	RL0	94
10.6.3	RL1	95
10.6.4	A0	95
10.6.5	A1	95
10.7	Free Parameters.	95
10.7.1	NICP, ICP	95
10.7.2	Fixed points.	95
10.7.3	Periodic solutions and rotations.	95
10.7.4	Folds and Hopf bifurcations.	96
10.7.5	Folds and period-doublings.	96
10.7.6	Boundary value problems.	96
10.7.7	Boundary value folds.	96
10.7.8	Optimization problems.	97
10.7.9	Internal free parameters.	97
10.7.10	Parameter overspecification.	97
10.8	Computation Constants.	98
10.8.1	ILP	98
10.8.2	ISP	98
10.8.3	ISW	98
10.8.4	MXBF	99
10.8.5	IRS	99
10.8.6	IPS	99
10.9	Output Control.	101
10.9.1	NPR	101
10.9.2	IID	101
10.9.3	IPLT	102
10.9.4	NUZR	102
10.10	Quick reference	104
11	Notes on Using AUTO.	105
11.1	Restrictions on the Use of PAR.	105
11.2	Efficiency.	105
11.3	Correctness of Results.	106
11.4	Bifurcation Points and Folds.	106
11.5	Floquet Multipliers.	106
11.6	Memory Requirements.	107
12	AUTO Demos : Tutorial.	108
12.1	Introduction.	109
12.2	cusps : A Tutorial Demo.	109
12.3	Copying the Demo Files.	109
12.4	Executing all Runs Automatically.	110
12.5	Plotting the Results with AUTO	112
12.6	Plotting the Results with AUTO in 3D.	113

12.7	Exporting the Results for different plotters.	113
12.8	ab : A Programmed Demo.	115
13	AUTO Demos : Fixed points.	116
13.1	enz : Stationary Solutions of an Enzyme Model.	116
13.2	dd2 : Fixed Points of a Discrete Dynamical System.	117
14	AUTO Demos : Periodic solutions.	118
14.1	lrz : The Lorenz Equations.	119
14.2	abc : The $A \rightarrow B \rightarrow C$ Reaction.	120
14.3	pp2 : A 2D Predator-Prey Model.	125
14.4	lor : Starting an Orbit from Numerical Data.	128
14.5	frc : A Periodically Forced System.	129
14.6	ppp : Continuation of Hopf Bifurcations.	130
14.7	plp : Fold Continuation for Periodic Solutions.	131
14.8	pp3 : Periodic Families and Loci of Hopf Points.	132
14.9	tor : Detection of Torus Bifurcations.	134
14.10	pen : Rotations of Coupled Pendula.	135
14.11	chu : A Non-Smooth System (Chua's Circuit).	137
14.12	phs : Effect of the Phase Condition.	138
14.13	ivp : Time Integration with Euler's Method.	139
15	AUTO Demos : BVP.	140
15.1	exp : Bratu's Equation.	140
15.2	int : Boundary and Integral Constraints.	141
15.3	bvp : A Nonlinear ODE Eigenvalue Problem.	142
15.4	lin : A Linear ODE Eigenvalue Problem.	143
15.5	non : A Non-Autonomous BVP.	144
15.6	kar : The Von Karman Swirling Flows.	145
15.7	spb : A Singularly-Perturbed BVP.	146
15.8	ezp : Complex Bifurcation in a BVP.	147
16	AUTO Demos : Parabolic PDEs.	148
16.1	pd1 : Stationary States (1D Problem).	149
16.2	pd2 : Stationary States (2D Problem).	150
16.3	wav : Periodic Waves.	151
16.4	brc : Chebyshev Collocation in Space.	152
16.5	brf : Finite Differences in Space.	153
16.6	bru : Euler Time Integration (the Brusselator).	154
17	AUTO Demos : Optimization.	155
17.1	opt : A Model Algebraic Optimization Problem.	156
17.2	ops : Optimization of Periodic Solutions.	157
17.3	obv : Optimization for a BVP.	161

18	AUTO Demos : Connecting orbits.	163
18.1	fsh : A Saddle-Node Connection.	164
18.2	nag : A Saddle-Saddle Connection.	165
18.3	stw : Continuation of Sharp Traveling Waves.	166
19	AUTO Demos : Miscellaneous.	168
19.1	pvl : Use of the Routine PVLS	169
19.2	ext : Spurious Solutions to BVB.	170
19.3	tim : A Test Problem for Timing AUTO	171
20	HomCont.	172
20.1	Introduction.	172
20.2	HomCont Files and Routines.	172
20.3	HomCont-Constants.	173
20.3.1	NUNSTAB	173
20.3.2	NSTAB	173
20.3.3	IEQUIB	173
20.3.4	ITWIST	174
20.3.5	ISTART	174
20.3.6	NREV, IREV	175
20.3.7	NFIXED, IFIXED	175
20.3.8	NPSI, IPSI	175
20.4	Restrictions on HomCont Constants.	175
20.5	Restrictions on the Use of PAR	176
20.6	Test Functions.	176
20.7	Starting Strategies.	177
20.8	Notes on Running HomCont Demos.	179
21	HomCont Demo : san.	181
21.1	Sandstede's Model.	181
21.2	Inclination Flip.	181
21.3	Non-orientable Resonant Eigenvalues.	183
21.4	Orbit Flip.	183
21.5	Detailed AUTO -Commands.	185
22	HomCont Demo : mtn.	188
22.1	A Predator-Prey Model with Immigration.	188
22.2	Continuation of Central Saddle-Node Homoclinics.	188
22.3	Switching between Saddle-Node and Saddle Homoclinic Orbits.	190
22.4	Three-Parameter Continuation.	191
22.5	Detailed AUTO -Commands.	192
23	HomCont Demo : kpr.	195
23.1	Koper's Extended Van der Pol Model.	195
23.2	The Primary Branch of Homoclinics.	195

23.3	More Accuracy and Saddle-Node Homoclinic Orbits.	199
23.4	Three-Parameter Continuation.	202
23.5	Detailed AUTO -Commands.	203
24	HomCont Demo : cir.	205
24.1	Electronic Circuit of Freire <i>et al.</i>	205
24.2	Detailed AUTO -Commands.	208
25	HomCont Demo : she.	209
25.1	A Heteroclinic Example.	209
25.2	Detailed AUTO -Commands.	211
26	HomCont Demo : rev.	213
26.1	A Reversible System.	213
26.2	An R_1 -Reversible Homoclinic Solution.	213
26.3	An R_2 -Reversible Homoclinic Solution.	214
26.4	Detailed AUTO -Commands.	218
27	HomCont Demo : Homoclinic branch switching.	219
27.1	Branch switching at an inclination flip in Sandstede's model.	219
27.2	Branch switching for a Shil'nikov type homoclinic orbit in the FitzHugh-Nagumo equations.	226
27.3	Branch switching to a 3-homoclinic orbit in a 5th-order Korteweg-De Vries model	229

Preface

This is a guide to the software package AUTO for continuation and bifurcation problems in ordinary differential equations. Earlier versions of AUTO were described in Doedel (1981), Doedel & Kernévez (1986*a*), Doedel & Wang (1995), Wang & Doedel (1995), Doedel, Champneys, Fairgrieve, Kuznetsov, Sandstede & Wang (1997), Doedel, Paffenroth, Champneys, Fairgrieve, Kuznetsov, Oldeman, Sandstede & Wang (2000). For a description of the basic algorithms see Doedel, Keller & Kernévez (1991*a*), Doedel, Keller & Kernévez (1991*b*). AUTO incorporates the HomCont algorithms of Champneys & Kuznetsov (1994), Champneys, Kuznetsov & Sandstede (1996) for the bifurcation analysis of homoclinic orbits. The Floquet multiplier algorithms were written by Fairgrieve (1994), Fairgrieve & Jepson (1991). A GUI was written by Wang (1994). The Python CLUI is the work of Randy Paffenroth.

Acknowledgments

The first author is much indebted to H. B. Keller of the California Institute of Technology for his inspiration, encouragement and support. He is also thankful to AUTO users and research collaborators who have directly or indirectly contributed to its development, in particular, Jean Pierre Kernévez, UTC, Compiègne, France; Don Aronson, University of Minnesota, Minneapolis; and Hans Othmer, University of Utah. Some material in this document related to the computation of connecting orbits was developed with Mark Friedman, University of Alabama, Huntsville. Also acknowledged is the work of Nguyen Thanh Long, Concordia University, Montreal, on the graphics program PLAUT. Special thanks are due to Sheila Shull, California Institute of Technology, for her cheerful assistance in the distribution of AUTO over a long period of time. Over the years, the development of AUTO has been supported by various agencies through the California Institute of Technology, and by research grants from NSERC (Canada).

The development of HomCont has benefitted from help and advice from, among others, W.-J. Beyn, Universität Bielefeld, M. J. Friedman, University of Alabama, A. Rucklidge, University of Cambridge, M. Koper, University of Utrecht, C. J. Budd, University of Bath, Fabio Dercole, Politecnico di Milano, and Frank Schilder, University of Surrey. Financial support for this collaboration was also received from the U.K. Engineering and Physical Science Research Council and the Nuffield Foundation.

License

AUTO is available under the terms of the BSD license:

Copyright © 1979–2007, E. J. Doedel, California Institute of Technology, and Concordia University. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

- Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
- Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer listed in this license in the documentation and/or other materials provided with the distribution.
- Neither the name of the copyright holders nor the names of its contributors may be used to endorse or promote products derived from this software without specific prior written permission.

THIS SOFTWARE IS PROVIDED BY THE COPYRIGHT HOLDERS AND CONTRIBUTORS "AS IS" AND ANY EXPRESS OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE COPYRIGHT OWNER OR CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Chapter 1

Installing AUTO.

1.1 Installation.

The AUTO file `auto07p-0.4.tar.gz` is available via <http://indy.cs.concordia.ca/auto>. Below it is assumed that you are using the Unix (e.g. *bash*) shell and that the file `auto07p-0.4.tar.gz` is in your main directory.

While in your main directory, enter the commands `gunzip auto07p-0.4.tar.gz`, followed by `tar xvfo auto07p-0.4.tar`. This will result in a directory `auto`, with one subdirectory, `auto/07p`. Type `cd auto/07p` to change directory to `auto/07p`. Then type `./configure`, to check your system for required compilers and libraries. Once the `configure` script has finished you may then type `make` to compile AUTO and its ancillary software. The `configure` script is designed to detect the details of your system which AUTO requires to compile successfully. If either the `configure` script or the `make` command should fail, you may assist the `configure` script by giving it various command line options. Please type `./configure --help` for more details. Upon compilation, you may type `make clean` to remove unnecessary files.

To run AUTO you need to set your environment variables correctly. Assuming AUTO is installed in your home directory, the following commands set your environment variables so that you will be able to run the AUTO commands, and may be placed into your `.login`, `.profile`, or `.cshrc` file, as appropriate. If you are using a `sh` compatible shell, such as `sh`, `bash`, `ksh`, or `ash` enter the command `source $HOME/auto/07p/cmds/auto.env.sh`. On the other hand, if you are using a `cs`h compatible shell, such as `cs`h or `tcsh`, enter the command `source $HOME/auto/07p/cmds/auto.env`.

The Graphical User Interface (GUI) requires the X-Window system and Motif or LessTif. Note that the GUI is not strictly necessary, since AUTO can be run very effectively using the Unix Command Language User Interface (CLUI). Moreover, long or complicated sequences of AUTO calculations can be programmed using the alternative Python CLUI. The GUI is not compiled by default. To compile AUTO with the GUI, type `./configure --enable-gui` and then `make` in the directory `auto/07p`.

The new graphic tool for AUTO data visualization, PLAUT04, is compiled by default, but depends on a few libraries that may not be in a standard installation of a typical Unix-like system. These libraries may be available as optional packages, though. Specifically, PLAUT04 requires at least either SGI Open Inventor 2.5.10 or Coin3D 2.2 with SoXt 1.1.0. It also needs

the libraries `simage 1.6` and `(Open)Motif` or `LessTif 2.0`. Any higher version numbers should also work. One can download SGI's implementation of the Open Inventor libraries from: <ftp://oss.sgi.com/projects/inventor/download/> Because SGI's implementation for Linux cannot show text correctly, we recommend that `Coin` is used instead of SGI's implementation. `Coin3D` can be downloaded from: <http://www.coin3d.org/download/> It is also available as a package for many Linux distributions. You will probably need to compile and install the `SoXt` and `simage` libraries from the source code available at the above web site. If, after that, `PLAUT04` still does not work then you might need to adjust the environment variable `LD_LIBRARY_PATH` to include the location of these libraries, for instance `/usr/local/lib`.

The `configure` script checks for these libraries and outputs a warning if any of these libraries cannot be found; if they are not available you can still compile all other components of `AUTO` using `make`.

For timing purposes, the file `auto/07p/src/autlib1.f` contains references to the function `etime`. Similarly, this file contains references to the subroutine `FLUSH` to avoid buffering of screen output when running `AUTO` from Python. These routines are not part of any Fortran standard but are common extensions. The `configure` script checks if your compiler supplies them. If they are not supplied then the dummy replacement routines in `src/compat.f` are used.

The PostScript conversion command `@ps` is compiled by default. Alternatively you can type `make` in the directory `auto/07p/tek2ps`. To generate the on-line manual, type `make` in `auto/07p/doc`.

To prepare `AUTO` for transfer to another machine, type `make superclean` in the directory `auto/07p` before creating the `tar`-file. This will remove all executable, object, and other non-essential files, and thereby reduce the size of the package.

Some `EISPACK` routines used by `AUTO` for computing eigenvalues and Floquet multipliers are included in the package (Smith, Boyle, Dongarra, Garbow, Ikebe, Klema & Moler (1976)).

1.1.1 Installation on Mac OS X

`AUTO` runs on Mac OS X using the above instructions provided that you have the development tools installed. You do not need to start an X server to run `AUTO`. To be able to plot in the Python CLUI, `AUTO` uses `pythonw` instead of `python`. This should happen automatically.

The old plotting tool `PLAUT` may not display fonts. To solve this issue you would need to obtain a different version of `xterm`; see <http://iparrizar.mnstate.edu/~juan/urania/2007/07/19/xterm-tektronix-emulation-broken-on-macos> or http://sourceforge.net/project/showfiles.php?group_id=21781. Apple is aware of this problem.

1.1.2 Installation on Windows

A native, light-weight solution for running `AUTO` on Windows is to use `GFortran`, `MSYS` (see <http://www.mingw.org>), combined with a native Win32 version of Python, obtained at <http://www.python.org>. To install this setup:

- Install Python from <http://www.python.org>.
- Install the minimal Unix-like environment `MSYS` from <http://www.mingw.org>. You do not need to install `MinGW` itself.

- Install GFortran from <http://gcc.gnu.org/wiki/GFortran>.
- Open a CMD.EXE window (click Start, Run, type `cmd` in the input box and press the Enter key), and run `sh` in the MSYS directory, for instance, using `c:\msys\1.0\bin\sh --login -i` (the Python CLUI does not work in the default MSYS shell environment accessed by its desktop icon).
- The `sh` command put you in a home directory, where you can unpack AUTO using `gunzip` and `tar`, as described above.
- Make sure that the `gfortran` and `python` binaries are in your `PATH`, and that their directories are at the front of it. You can do this, for instance, using the shell command `export PATH="/c/Python24:/c/Program Files/gfortran/bin:$PATH"` . You can also inspect, edit and then source the file `auto/07p/cmds/auto.env.sh` to achieve this.
- Now you should be able to run `configure`, `make`, and run AUTO as shown above.

Alternatively, AUTO runs on Windows as above using the Unix-like environment Cygwin (see <http://www.cygwin.com>). You should have at least the default installation, `gcc`, `gfortran`, `make`, and `python` installed. The 2D-plot command is not as responsive in Cygwin Python as in the above native setup, however.

In both cases, an X server is not necessary, unless you wish to try to run PLAUT, PLAUT04, or the GUI.

1.2 Restrictions on Problem Size.

There are size restrictions in the file `auto/07p/include/auto.h` on the following AUTO-constants: the effective number of equation parameters `NPAR`, and the number of stored branch points `NBIF` for algebraic problems, See Chapter 10 for the significance these constants. Their maxima are denoted by the corresponding constant followed by `anX`. For example, `NPARX` in `auto.h` denotes the maximum value of `NPAR`. If any of these maxima is exceeded in an AUTO-run then a message will be printed. The exception is the the maximum value of `NPAR`, which, if exceeded, may lead to unreported errors. Upon installation `NPARX=36`; it should never be decreased below that value; see also Section 11.1. Size restrictions can be changed by editing `auto.h`. This must be followed by recompilation by typing `make` in directory `auto/07p/src`.

Note that in certain cases the *effective dimension* may be greater than the user dimension. For example, for the continuation of folds, the effective dimension is `2NDIM+1` for algebraic equations, and `2NDIM` for ordinary differential equations, respectively. Similarly, for the continuation of Hopf bifurcations, the effective dimension is `3NDIM+2`.

1.3 Compatibility with Earlier Versions.

Unlike earlier versions, AUTO can no longer be compiled using a pure Fortran 77 compiler, but you need at least a Fortran 90 compiler. A free Fortran 95 compiler, GFortran, is shipped with most recent Linux distributions, or can be obtained at <http://gcc.gnu.org/wiki/GFortran>,

which contains binaries for Linux, Mac OS X and Windows. AUTO was also tested with the free compiler **g95**, and there exist various, untested, commercial Fortran 9x compilers as well.

The AUTO input files are now called **c.xxx** (the constants file), and **h.xxx** (the HomCont constants file, only used with HomCont); the output files are called **b.xxx** (the bifurcation-diagram-file), **s.xxx** (the solution-file), and **d.xxx** (the diagnostics-file). There are also minor changes in the formatting of these files compared to recent versions of AUTO, such as AUTO97 and AUTO2000. The main change compared to AUTO97 is that there is now a programmable Python CLUI.

When upgrading from AUTO2000, you can continue to use equations-files written in C. However, there is now a strict difference between indexing of the array **par[]** in the C file and the references to it using **PAR()** in constants files and output, using **par[i]=PAR(I+1)**. In practise this means that you do not have to change the C file, but need to add 1 to all parameter indices in the constant files, namely **ICP(I)**, **THL(I)**, and **UZR(I)**. For example, the period is referenced by **par[10]** in the C file, but by **PAR(11)** in the constants file. Equation files written in C are used in the homoclinic branch switching demo in Chapter 27.

1.4 Parallel Version.

AUTO contains code which allows it to run in on parallel computers. Namely, it can use either OpenMP to run most of its code in parallel on shared-memory multi-processors, or the MPI message passing library. When the **configure** script is run it will try to detect if the Fortran compiler supports OpenMP; examples are Gfortran 4.2 and the Intel Fortran Compiler. If it is successful the necessary compiler flags are used to enable OpenMP in AUTO. To force the **configure** script not to use OpenMP, one may type **./configure --without-openmp**, and then type **make**. On the other hand, unless there is some particular difficulty, we recommend that that the **configure** script be used without arguments, since the parallel version of AUTO may easily be controlled, and even run in a serial mode, through the use of the environment variable **OMP_NUM_THREADS**.

For example, to run the AUTO executable **auto.exe** in serial mode you just type **export OMP_NUM_THREADS=1**. To run the same command in parallel on 4 processors you type **export OMP_NUM_THREADS=4**. Without any **OMP_NUM_THREADS** set the number of processors that AUTO will use can be equal to the actual number of processors on the system, or can be equal to one; this is system-dependent.

The MPI message passing library is not used by default. You can enable it by typing **./configure --with-mpi**. If OpenMP and MPI are both used then AUTO uses mixed mode, with MPI parallelisation occurring at the top level.

Running the MPI version is somewhat more complex because of the fact that MPI normally uses some external program for starting the computational processes. The exact name and command line options of this external program depends on your MPI installation. A common name for this MPI external program is **mpirun**, and a common command line option which defines the number of computational processes is **-np**. Accordingly, if you wanted to run the MPI version of AUTO on four processors, with the above external program, you would type **mpirun -np 4 file.exe**. Please see your local MPI documentation for more detail.

Chapter 2

Overview of Capabilities.

2.1 Summary.

AUTO can do a limited bifurcation analysis of algebraic systems

$$f(u, p) = 0, \quad f(\cdot, \cdot), u \in \mathbb{R}^n. \quad (2.1)$$

The main algorithms in AUTO, however, are aimed at the continuation of solutions of systems of ordinary differential equation (ODEs) of the form

$$u'(t) = f(u(t), p), \quad f(\cdot, \cdot), u(\cdot) \in \mathbb{R}^n, \quad (2.2)$$

subject to boundary (including initial) conditions and integral constraints. Above, p denotes one or more free parameters,

These boundary value algorithms also allow AUTO to do certain stationary solution and wave calculations for the partial differential equation (PDE)

$$u_t = Du_{xx} + f(u, p), \quad f(\cdot, \cdot), u(\cdot) \in \mathbb{R}^n, \quad (2.3)$$

where D denotes a diagonal matrix of diffusion constants.

The basic algorithms used in AUTO, as well as related algorithms, can be found in Keller (1977), Keller (1986), Doedel, Keller & Kernévez (1991a), Doedel, Keller & Kernévez (1991b).

Below, the basic capabilities of AUTO are specified in more detail. Some representative demos are also indicated.

2.2 Algebraic Systems.

Specifically, for (2.1) AUTO can :

- Compute solution families.
(Demo `ab`; Run 1.)
- Locate branch points, continue these in two parameters, and automatically compute bifurcating families.
(Demos `pp2`; Run 1 and `apnf`.)

- Locate Hopf bifurcation points and continue these in two parameters.
(Demo **ab**; Runs 1 and 5.)
- Locate folds (limit points) and continue these in two parameters.
(Demo **ab**; Runs 1,3,4.)
- Do each of the above for fixed points of the discrete dynamical system $u^{(k+1)} = f(u^{(k)}, p)$
(Demo **dd2**.)
- Find extrema of an objective function along solution families and successively continue such extrema in more parameters.
(Demo **opt**.)

2.3 Ordinary Differential Equations.

For the ODE (2.2) the program can :

- Compute families of stable and unstable periodic solutions and compute the Floquet multipliers, that determine stability, along these families. Starting data for the computation of periodic orbits are generated automatically at Hopf bifurcation points.
(Demo **ab**; Run 2.)
- Locate folds, branch points, period doubling bifurcations, and bifurcations to tori, along families of periodic solutions. Branch switching is possible at branch points and at period doubling bifurcations.
(Demos **tor**, **lor**.)
- Continue folds and period-doubling bifurcations, in two parameters.
(Demos **plp**, **pp3**.) The continuation of orbits of fixed period is also possible. This is the simplest way to compute curves of homoclinic orbits, if the period is sufficiently large.
(Demo **pp2**.)
- Do each of the above for *rotations*, i.e., when some of the solution components are periodic modulo a phase gain of a multiple of 2π .
(Demo **pen**.)
- Follow curves of homoclinic orbits and detect and continue various codimension-2 bifurcations, using the HomCont algorithms of Champneys & Kuznetsov (1994), Champneys, Kuznetsov & Sandstede (1996).
(Demos **san**, **mnt**, **kpr**, **cir**, **she**, **rev**.)
- Locate extrema of an integral objective functional along a family of periodic solutions and successively continue such extrema in more parameters.
(Demo **ops**.)

- Compute curves of solutions to (2.2) on $[0, 1]$, subject to general nonlinear boundary and integral conditions. The boundary conditions need not be separated, i.e., they may involve both $u(0)$ and $u(1)$ simultaneously. The side conditions may also depend on parameters. The number of boundary conditions plus the number of integral conditions need not equal the dimension of the ODE, provided there is a corresponding number of additional parameter variables.
(Demos `exp`, `int`.)
- Determine folds and branch points along solution families to the above boundary value problem. Branch switching is possible at branch points. Curves of folds can be computed in two parameters.
(Demos `bvp`, `int`.)

2.4 Parabolic PDEs.

For (2.3) the program can :

- Trace out families of spatially homogeneous solutions. This amounts to a bifurcation analysis of the algebraic system (2.1). However, AUTO uses a related system instead, in order to enable the detection of bifurcations to wave train solutions of given wave speed. More precisely, bifurcations to wave trains are detected as Hopf bifurcations along fixed point families of the related ODE

$$\begin{aligned} u'(z) &= v(z), \\ v'(z) &= -D^{-1}[c v(z) + f(u(z), p)], \end{aligned} \tag{2.4}$$

where $z = x - ct$, with the wave speed c specified by the user.

(Demo `wav`; Run 2.)

- Trace out families of periodic wave solutions to (2.3) that emanate from a Hopf bifurcation point of Equation 2.4. The wave speed c is fixed along such a family, but the wave length L , i.e., the period of periodic solutions to (2.4), will normally vary. If the wave length L becomes large, i.e., if a homoclinic orbit of Equation 2.4 is approached, then the wave tends to a solitary wave solution of (2.3).
(Demo `wav`; Run 3.)
- Trace out families of waves of fixed wave length L in two parameters. The wave speed c may be chosen as one of these parameters. If L is large then such a continuation gives a family of approximate solitary wave solutions to (2.3).
(Demo `wav`; Run 4.)
- Do time evolution calculations for (2.3), given periodic initial data on the interval $[0, L]$. The initial data must be specified on $[0, 1]$ and L must be set separately because of internal scaling. The initial data may be given analytically or obtained from a previous computation of wave trains, solitary waves, or from a previous evolution calculation. Conversely, if an evolution calculation results in a stationary wave then this wave can be used as starting

data for a wave continuation calculation.

(Demo `wav`; Run 5.)

- Do time evolution calculations for (2.3) subject to user-specified boundary conditions. As above, the initial data must be specified on $[0, 1]$ and the space interval length L must be specified separately. Time evolution computations of (2.3) are adaptive in space and in time. Discretization in time is not very accurate : only implicit Euler. Indeed, time integration of (2.3) has only been included as a convenience and it is not very efficient. (Demos `pd1`, `pd2`.)
- Compute curves of stationary solutions to (2.3) subject to user-specified boundary conditions. The initial data may be given analytically, obtained from a previous stationary solution computation, or from a time evolution calculation. (Demos `pd1`, `pd2`.)

In connection with periodic waves, note that (2.4) is just a special case of (2.2) and that its fixed point analysis is a special case of (2.1). One advantage of the built-in capacity of AUTO to deal with problem (2.3) is that the user need only specify f , D , and c . Another advantage is the compatibility of output data for restart purposes. This allows switching back and forth between evolution calculations and wave computations.

2.5 Discretization.

AUTO discretizes ODE boundary value problems (which includes periodic solutions) by the method of orthogonal collocation using piecewise polynomials with 2-7 collocation points per mesh interval (de Boor & Swartz (1973)). The mesh automatically adapts to the solution to equidistribute the local discretization error (Russell & Christiansen (1978)). The number of mesh intervals and the number of collocation points remain constant during any given run, although they may be changed at restart points. The implementation is AUTO-specific. In particular, the choice of local polynomial basis and the algorithm for solving the linearized collocation systems were specifically designed for use in numerical bifurcation analysis.

Chapter 3

User-Supplied Files.

The user must prepare the two files described below. This can be done with the GUI described in Chapter 9, or independently.

3.1 The Equations-File `xxx.f90`, or `xxx.f`, or `xxx.c`

A source file `xxx.f90` containing the Fortran routines `FUNC`, `STPNT`, `BCND`, `ICND`, `FOPT`, and `PVLS`. Here `xxx` stands for a user-selected name. If any of these routines is irrelevant to the problem then its body need not be completed. Examples are in `auto/07p/demos`, where, e.g., the file `ab/ab.f` defines a two-dimensional dynamical system, and the file `exp/exp.f` defines a boundary value problem. The simplest way to create a new equations-file is to copy an appropriate demo file. For a fully documented equations-file see `auto/07p/demos/cusp/cusp.f90` or `auto/07p/gui/aut.f`. In GUI mode, this file can be directly loaded with the GUI-button *Equations/New*; see Section 9.2.

The equations-file can either be written in fixed-form (old-style) Fortran (`.f`), free-form Fortran (`.f90`) or in C (`.c`).

3.2 The Constants-File `c.xxx`

AUTO-constants for `xxx.{f,f90,c}` are normally expected in a corresponding file `c.xxx`. Specific examples include `ab/c.ab` and `exp/c.exp` in `auto/07p/demos`. See Chapter 10 for the significance of each constant.

3.3 User-Supplied Routines.

The purpose of each of the user-supplied routines in the file `xxx.f90`, `f` is described below.

- **FUNC** : defines the function $f(u, p)$ in (2.1), (2.2), or (2.3).
- **STPNT** : This routine is called only if `IRS=0` (see Section 10.8.5 for `IRS`), which typically is the case for the first run. It defines a starting solution (u, p) of (2.1) or (2.2). The starting solution should not be a branch point.
(Demos `ab`, `exp`, `frc`, `lor`.)
- **BCND** : A routine `BCND` that defines the boundary conditions.
(Demo `exp`, `kar`.)
- **ICND** : A routine `ICND` that defines the integral conditions.
(Demos `int`, `lin`.)
- **FOPT** : A routine `FOPT` that defines the objective functional.
(Demos `opt`, `ops`.)
- **PVLS** : A routine `PVLS` for defining “solution measures”.
(Demo `pv1`.)

In a C language equation file, these routines are written using lowercase letters; with Fortran you can use any case.

3.4 User-Supplied Derivatives.

If `AUTO`-constant `JAC` equals 0 then derivatives need not be specified in `FUNC`, `BCND`, `ICND`, and `FOPT`; see Section `sec:JAC`. If `JAC=1` then derivatives must be given. This may be necessary for sensitive problems, and is recommended for computations in which `AUTO` generates an extended system. Examples of user-supplied derivatives can be found in demos `dd2`, `int`, `plp`, `opt`, and `ops`.

Chapter 4

Running AUTO using Python Commands.

4.1 Typographical Conventions

This chapter uses the following conventions. All code examples will be in in the following font.

```
AUTO> copydemo("ab")
Copying demo ab ... done
```

To distinguish commands which are typed to the Unix shell from those which are typed to the AUTO command line user interface (CLUI) we will use the following two prompts.

>	Commands which follow this prompt are for the Unix shell.
AUTO>	Commands which follow this prompt are for the AUTO CLUI.

4.2 General Overview.

The AUTO command line user interface (CLUI) is similar to the command language described in Section 5 in that it facilitates the interactive creating and editing of equations-files and constants-files. It differs from the other command language in that it is based on the object-oriented scripting language Python (see Lutz (1996)) and provides extensive programming capabilities. This chapter will provide documentation for the AUTO CLUI commands, but is not intended as a tutorial for the Python language. We will attempt to make this chapter self contained by describing all Python constructs that we use in the examples, but for more extensive documentation on the Python language, including tutorials and pointers to further documentation, please see Lutz (1996) or the web page <http://www.python.org> which contains an excellent tutorial at <http://www.python.org/doc/current/tut/tut.html>.

To use the CLUI for a new equation, change to an empty directory. For an existing equations-file, change to its directory. (*Do not activate the CLUI in the directory `auto/07p` or in any of its subdirectories.*) Then type

auto.

```
> auto
Initializing
Python 1.5.2 (1, Feb  1 2000, 16:32:16) [GCC egcs-2.91.66 19990314/Linux
Copyright 1991-1995 Stichting Mathematisch Centrum, Amsterdam
(AUTOInteractiveConsole)
AUTO>
```

Figure 4.1: Typing `auto` at the Unix shell prompt starts the AUTO CLUI.

If your command search path has been correctly set (see Section 1.1), this command will start the AUTO CLUI interactive interpreter and provide you with the AUTO CLUI prompt.

In addition to the examples in the following sections there are several example scripts which can be found in `auto/07p/demos/python` and are listed in Table 4.1. These scripts are fully annotated and provide good examples of how AUTO CLUI scripts are written. The scripts in `auto/07p/demos/python/n-body` are especially lucid examples and perform various related parts of a calculation involving the gravitational N-body problem. Scripts which end in the suffix `.auto` are called “basic” scripts and can be run by typing `auto scriptname.auto`. The scripts shown in Section 4.3 and Section 4.5 are examples of basic scripts. Scripts which end in the suffix `.xauto` are called “expert” scripts and can be run by typing `autox scriptname.xauto`. More information on expert scripts can be found in Section 4.6. See the `README` file in that directory for more information.

4.3 First Example

We begin with a simple example of the AUTO CLUI. In this example we copy the `ab` demo from the AUTO installation directory and run it. For more information on the `ab` demo see Section 12.8. The commands listed in Table 4.2 will copy the demo files to your work directory and run the first part of the demo. The results of running these commands are shown in Figure 4.2.

Let us examine more closely what action each of the commands performs. First, `copydemo('ab')` (Section 4.14.7 in the reference) copies the files in `$AUTO.DIR/demo/ab` into the work directory.

Next, `load(equation='ab')` (Section 4.14.34 in the reference) informs the AUTO CLUI that the name of the user defined function file is `ab.f`. The command `load` is one of the most commonly used commands in the AUTO CLUI, since it reads and parses the user files which are manipulated by other commands. The AUTO CLUI stores this setting until it is changed by a command, such as another `load` command. The idea of storing information is one of the ideas that sets the CLUI apart from the command language described in Section 5.

Next, `load(constants='ab.1')` parses the AUTO constants file `c.ab.1` and reads it into memory. Note that *changes to the file `c.ab.1` after it has been loaded in will not be used by AUTO unless it is loaded in again after the changes are made.*

Finally, `run()` (Section 4.14.31 in the reference) uses the user defined functions loaded by the `load(equation='ab')` command, and the AUTO constants loaded by the `load(constants='ab.1')` to run AUTO.

Script	Description
demo1.auto	The demo script from Section 4.3.
demo2.auto	The demo script from Section 4.5.
userScript.xauto	The expert demo script from Figure 4.12.
userScript.py	The loadable expert demo script from Figure 4.13.
fullTest.auto	A script which uses the entire AUTO command set, except for the plotting commands.
plotter.auto	A demonstration of some of the plotting capabilities of AUTO .
fullTest.auto	A script which implements the tutorial from Section 12.8.
n-body/compute_lagrange_points_family.auto	A basic script which computes and plots all of the “Lagrange points” as a function of the ratio of the masses of the two planets.
n-body/compute_lagrange_points_0.5.auto	A basic script which computes all of the “Lagrange points” for the case where the masses of the two planets are equal, and saves the data.
n-body/compute_periodic_family.xauto	An expert script which starts at a “Lagrange point” computed by compute_lagrange_points_0.5.auto and continues in the ratio of the masses until a specified mass ratio is reached. It then computes a family of periodic orbits for each pair of purely complex eigenvalues.
n-body/to_matlab.xauto	A script which takes a set of AUTO data files and creates a set of files formatted for importing into Matlab for either plotting or further calculations.

Table 4.1: The various demonstration scripts for the AUTO CLUI.

Unix-COMMAND	ACTION
<code>auto</code>	start the AUTO CLUI
AUTO CLUI COMMAND	ACTION
<code>copydemo('ab')</code>	copy the demo files to the work directory
<code>load(equation='ab')</code>	load the filename <code>ab.f</code> into memory
<code>load(constants='ab.1')</code>	load the contents of the file <code>c.ab.1</code> into memory
<code>run()</code>	run AUTO with the current set of files

Table 4.2: Running the demo `ab` files.


```

> auto
Python 2.3.5 (#2, Jun 13 2006, 23:12:55)
[GCC 4.1.2 20060613 (prerelease) (Debian 4.1.1-4)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
(AUTOInteractiveConsole)
AUTO> copydemo('ab')
Copying demo ab ... done
AUTO> load(equation='ab')
Runner configured
AUTO> load(constants='ab')
Runner configured
AUTO> run()
gfortran -O -c ab.f -o ab.o
gfortran -O ab.o -o ab.exe /home/enbeo/auto/07pp/lib/*.o
Starting ab ...

  BR    PT  TY  LAB    PAR(2)    L2-NORM    U(1)    U(2)
  1      1  EP   1    8.00000E+00  0.00000E+00  0.00000E+00  0.00000E+00
  1     31  UZ   2    1.40000E+01  0.00000E+00  0.00000E+00  0.00000E+00
  1     36  UZ   3    1.50000E+01  0.00000E+00  0.00000E+00  0.00000E+00
  1     41  UZ   4    1.60000E+01  0.00000E+00  0.00000E+00  0.00000E+00
  1     46  UZ   5    1.70000E+01  0.00000E+00  0.00000E+00  0.00000E+00
  1     51  EP   6    1.80000E+01  0.00000E+00  0.00000E+00  0.00000E+00

Total Time    0.240E-01
ab ... done
AUTO>

```

Figure 4.2: Typing `auto` at the Unix shell prompt starts the AUTO CLUI. The rest of the commands are interpreted by the AUTO CLUI.

Figure 4.2 showed two of the file types that the `load` command can read into memory, namely the user defined function file and the AUTO constants file (Section 3). There are two other files types that can be read in using the `load` command, and they are the restart solution file (Section 6) and the HomCont parameter file (Section 20.2).

Note that the name given to the `load` command is not the same as the filename which is read in, for example `load(constants='ab.1')` reads in the file `c.ab.1`. This difference is a result of the automatic transformation of the filenames by the AUTO CLUI into the standard names used by AUTO . The standard filename transformations are show in Table 4.3.

Long name	Short name	Name entered	Transformed file name
equation	e	foo	foo.c
constants	c	foo	c.foo
solution	s	foo	s.foo
bifurcationDiagram	b	foo	b.foo
diagnostics	d	foo	d.foo
homcont	h	foo	h.foo

Table 4.3: This table shows the standard AUTO CLUI filename translations. In `load` and `run` commands either the long name or the short name may be used for loading the appropriate files.

Since the `load` command is so common, there are various shorthand versions of it. First, there are short versions of the various arguments as shown in Table 4.3. For example, the command `load(constants='ab.1')` can be shortened to `load(c='ab.1')`. Next, several different files may be loaded at once using the same `load` command. For example, the two commands in Figure 4.3 have the same effect as the single command in Figure 4.4.

```
AUTO> load(e='ab')
Runner configured
AUTO> load(c='ab.1')
Runner configured
```

Figure 4.3: Loading two files individually.

```
AUTO> load(e='ab',c='ab.1')
Runner configured
```

Figure 4.4: Loading two files at the same time.

Also, since it is common that several files will be loaded that have the same base name `load('ab')` performs the same action as `load(e='ab',c='ab',s='ab',h='ab')`. Note, for the command `load('ab')` it is only required that `ab.f` and `c.ab` exist; `s.ab` and `h.ab` are optional, and if they do not exist, no error message will be given.

4.4 Scripting

Section 4.3 showed commands being interactively entered at the AUTO CLUI prompt, but since the AUTO CLUI is based on Python one has the ability to write scripts for performing sequences of commands automatically. A Python script is very similar to the interactive mode shown in Section 4.3 except that the commands are placed in a file and read all at once. For example, if the commands from Figure 4.2 were placed into the file `demo1.auto`, in the format shown in Figure 4.5, then the commands could be run all at once by typing `auto demo1.auto`. See Figure 4.6 for the full output.

```
copydemo('ab')
load(equation='ab')
load(constants='ab.1')
run()
```

Figure 4.5: The commands from Figure 4.2 and they would appear in a AUTO CLUI script file. The source for this script can be found in `$AUTO_DIR/demos/python/demo1.auto`.

4.5 Second Example

In Section 4.3 we showed a very simple AUTO CLUI script, in this Section we will describe a more complex example, which introduces several new AUTO CLUI commands as well as some basic Python constructs for conditionals and looping. We will not provide an exhaustive reference for the Python language, but only the very basics. For more extensive documentation we refer the reader to Lutz (1996) or the web page <http://www.python.org>. In this section we will describe each line of the script in detail, and the full text of the script is in Figure 4.7.

The script begins with a section, extracted into Figure 4.8, which performs a task identical to that shown in Figure 4.2 except that the shorthand discussed in Section 4.3 is used for the `ld` command.

The next section of the script, extracted into Figure 4.9, introduces three new AUTO CLUI commands. First, `sv('bvp')` (Section 4.14.6 in the reference) saves the results of the AUTO run into files using the base name `bvp` and the filename extensions in Table 4.3. For example, in this case the bifurcation diagram file `fort.7` will be saved as `b.bvp`, the solution file `fort.8` will be saved as `s.bvp`, and the diagnostics file `fort.9` will be saved as `d.bvp`. Next, `ld(s='bvp')` loads the solution file `s.bvp` into memory so that it can be used by AUTO for further calculations.

Up to this point all of the commands presented have had analogs in the command language discussed in Section 5, and the AUTO CLUI has been designed in this way to make it easy for users to migrate from the old command language to the AUTO CLUI. The next command, namely `data = sl('bvp')` (Section 4.14.19 in the reference) is the first command which has no analog in the old command language. The command `sl('bvp')` parses the file `s.bvp` and returns a python object which encapsulates the information contained in the file and presents it to the user in an easy to use format. Accordingly, the command `data = sl('bvp')` stores this easy

```

> cat demo1.auto
copydemo('ab')
load(equation='ab')
load(constants='ab.1')
run()

> auto demo1.auto
Initializing
Copying demo ab ... done
Runner configured
Runner configured
gcc -O -DPTHREADS -O -I/home/amavisitors/redrod/src/auto/2000/include -c ab.c
gcc -O ab.o -o ab.exe /home/amavisitors/redrod/src/auto/2000/lib/*.o -lpthread
-L/home/amavisitors/redrod/src/auto/2000/lib -lauto_f2c -lm
Starting ab ...

      1      1  EP   1  0.000000E+00  0.000000E+00  0.000000E+00  0.000000E+00
      1     33  LP   2  1.057390E-01  1.484391E+00  3.110230E-01  1.451441E+00
      1     70  LP   3  8.893185E-02  3.288241E+00  6.889822E-01  3.215250E+00
      1     90  HB   4  1.308998E-01  4.271867E+00  8.950803E-01  4.177042E+00
      1     92  EP   5  1.512417E-01  4.369748E+00  9.155894E-01  4.272750E+00

Total Time      8.740E-02
ab ... done
>

```

Figure 4.6: This Figure starts by listing the contents of the `demo1.auto` file using the Unix `cat` command. The file is then run through the AUTO CLUI by typing `auto demo1.auto` and the output is shown.

to use representation of the object in the Python variable `data`. Note, variables in Python are different from those in languages such as C in that their type does not have to be declared before they are created. Finally, `ch("NTST",50)` (Section 4.14.32 in the reference) changes the `NTST` value to 50 (see Section 10.2.1). To be precise, the command `ch("NTST",50)` only modifies the “in memory” version of the AUTO constants created by the `ld('bvp')` command. The original file `c.bvp` is *not* modified.

The next section of the script, extracted into Figure 4.10, shows as example of looping and conditionals in an AUTO CLUI script. The first line `for solution in data:` is the Python syntax for loops. The `data` variable was defined in Figure 4.9 to be the parsed version of an AUTO `fort.8` file, and accordingly contains a list of the solutions from the `fort.8` file. The command `for solution in data:` is used to loop over all solutions in the `data` variable by setting the variable `solution` to be one of the solutions in `data` and then calling the rest of the code in the block.

Python differs from most other computer languages in that blocks of code are not defined by

```

copydemo('bvp')

ld('bvp')
run()
sv('bvp')
ld(s='bvp')
data = sl('bvp')
ch("NTST",50)
for solution in data:
    if solution["Type name"] == "BP":
        ch("IRS", solution["Label"])
        ch("ISW", -1)
        # Compute forward
        run()
        ap('bvp')
        # Compute back
        ch("DS",-pr("DS"))
        run()
        ap('bvp')

plot('bvp')
wait()

```

Figure 4.7: This Figure shows a more complex AUTO CLUI script. The source for this script can be found in `$AUTO_DIR/demos/python/demo2.auto`.

```

copydemo('bvp')

ld('bvp')
run()

```

Figure 4.8: The first part of the complex AUTO CLUI script.

```

sv('bvp')
ld(s='bvp')
data = sl('bvp')
ch("NTST",50)

```

Figure 4.9: The second part of the complex AUTO CLUI script.

some delimiter, such as `{}` in C, but by indentation. In Figure 4.7 the commands `plot('bvp')` and `wait()` are not part of the loop, because they are indented differently. This can be confusing first time users of Python, but it has the advantage that the code is forced to have a consistent indentation style.

The next command in the script, `if solution["Type name"] == "BP":` is a Python conditional. It examines the contents of the variable `solution` (which is one of the entries in the array of solutions `data`) and checks to see if the condition `solution["Type name"] == "BP"` holds. For parsed `fort.8` files `Type name` BP corresponds to a bifurcation point. Accordingly, the function of this loop and conditional is to examine every solution in the `fort.8` file and run the following commands if the solution is a bifurcation point.

The next line is `ch("IRS", solution["Label"])` which changes the “in memory” version of the AUTO constants file to set IRS (see Section 10.8.5) equal to the label of the bifurcation point. We then use `ch("ISW", -1)` to change the AUTO constant ISW to -1, which indicates a branch switch (see Section 10.8.3).

We then use a `run()` command to perform the calculation of the bifurcating branch and then append the data to the `s.bvp`, `b.bvp`, and `d.bvp` files with the `ap('bvp')` command (Section 4.14.1 in the reference). In addition, as can be seen in Figure 4.10, the `#` character is the Python comment character. When the Python interpreter encounters a `#` character it ignores everything from that character to the end of the line.

Finally, we use `ch("DS",-pr("DS"))` to change the AUTO initial step size from positive to negative, which allows us to compute the bifurcating branch in the other direction (see Section 10.5.1). Running the AUTO calculation with the `run()` command and appending the data the appropriate files with the `ap('bvp')` command completes the body of the loop.

```
for solution in data:
    if solution["Type name"] == "BP":
        ch("IRS", solution["Label"])
        ch("ISW", -1)
        # Compute forward
        run()
        ap('bvp')
        # Compute back
        ch("DS",-pr("DS"))
        run()
        ap('bvp')
```

Figure 4.10: The second part of the complex AUTO CLUI script.

Now that the section of script shown in Figure 4.10 has finished computing the bifurcation diagram, the command `plot('bvp')` brings up a plotting window (Section 4.14.20 in the reference), and the command `wait()` causes the AUTO CLUI to wait for input. You may now exit the AUTO CLUI by pressing any key in the window in which you started the AUTO CLUI.

For convenience, some of these commands have shorter forms. For instance, you can directly specify a change of NTST in the `ld` and `run` commands, by giving `NTST=xxx` as an extra pa-

parameter, as well as the special `DS='-'` notation to quickly revert the direction (the sign of `DS`). The command `run` also accepts the files to save or append to using `sv=` and `ap=`, respectively. Another helpful function is `splabs`, that return a list of labels for a given solution file. A shorter version of the complex AUTO CLUI script is given in Figure 4.11.

```
copydemo('bvp')

#run bvp.f with c.bvp (first parameter), save in {b,s}.bvp (second parameter)
r('bvp','bvp')
ld(s='bvp',NTST=50)
for solution in splabs('bvp','BP'):
    # Compute forward
    r(IRS=solution,ISW=-1,ap='bvp')
    # Compute back
    r(DS='- ',ap='bvp')

pl('bvp')
wait()
```

Figure 4.11: This Figure shows a shorter version of the more complex AUTO CLUI script given above. The source for this script can be found in `$AUTO_DIR/demos/python/demo3.auto`.

4.6 Extending the AUTO CLUI

The code in Figure 4.7 performed a very useful and common procedure, it started an AUTO calculation and performed additional continuations at every point which AUTO detected as a bifurcation. Unfortunately, the script as written can only be used for the `bvp` demo. In this section we will generalize the script in Figure 4.7 for use with any demo, and demonstrate how it can be imported back into the interactive mode to create a new command for the AUTO CLUI. Several examples of such “expert” scripts can be found in `auto/07p/demos/python/n-body`.

Just as loops and conditionals can be used in Python, one can also define functions. For example, Figure 4.12 is a functional version of script from Figure 4.7. The changes are actually quite minor. The first line, `from AUTOclui import *`, includes the definitions of the AUTO CLUI commands, and must be included in all AUTO CLUI scripts which define functions. The next line, `def myRun(demo):`, begins the function definition, and creates a function named `myRun` which takes one argument `demo`. The rest of the script is the same except that it has been indented to indicate that it is part of the function definition, and all occurrences of string `'bvp'` have been replaced with the variable `demo`. Finally we have added a line `myRun('bvp')` which actually calls the function we have created and runs the same computation as the original script.

While the script in Figure 4.12 is only slightly different then the one showed in Figure 4.7 it is much more powerful. Not only can it be used as a script for running any demo by modifying the last line, it can be read back into the interactive mode of the AUTO CLUI and used to create

```

from AUTOclui import *

def myRun(demo):

    copydemo(demo)

    ld(demo)
    run()
    sv(demo)
    ld(s=demo)
    data = sl(demo)
    ch("NTST",50)
    for solution in data:
        if solution["Type name"] == "BP":
            ch("IRS", solution["Label"])
            ch("ISW", -1)
            # Compute forward
            run()
            ap(demo)
            # Compute back
            ch("DS",-pr("DS"))
            run()
            ap(demo)

    plot(demo)
    wait()

myRun('bvp')

```

Figure 4.12: This Figure shows a complex AUTO CLUI script written as a function. The source for this script can be found in `$AUTO_DIR/demos/python/userScript.xauto`.

a new command, as in Figure 4.13. First, we create a file called `userScript.py` which contains the script from Figure 4.12, with one minor modification. We want the function only to run when we use it interactively, not when the file `userScript.py` is read in, so we remove the last line where the function is called. We start the AUTO CLUI with the Unix command `auto`, and once the AUTO CLUI is running we use the command `from userScript import *`, to import the file `userScript.py` into the AUTO CLUI. The `import` command makes all functions in that file available for our use (in this case `myRun` is the only one). It is important to note that `from userScript import *` does *not* use the `.py` extension on the file name. After importing our new function, we may use it just like any other function in the AUTO CLUI, for example by typing `myRun('bvp')`.

4.7 Bifurcation Diagram Files

Using the `commandParseDiagramFile` command (Section 4.14.18 in the reference) the user can parse and read into memory an AUTO bifurcation diagram file. For example, the command `commandParseDiagramFile('ab')` would parse the file `b.ab` (if you are using the standard file-name translations from Table 4.3) and return an object which encapsulates the bifurcation diagram in an easy to use form.

The object returned by the `commandParseDiagramFile` is a list of all of the solutions in the appropriate bifurcation diagram file, and each solution is a Python dictionary with entries for each piece of data for the solution. For example, the sequence of commands in Figure 4.14, prints out the label of the first solution in a bifurcation diagram. The queriable parts of the object are listed in Table 4.4.

The individual elements of the array may be accessed in two ways, either by index of the solution using the `[]` syntax or by label number using the `()` syntax. For example, assume that the parsed object is contained in a variable `data`. The first solution may be accessed using the command `data[0]`, while the solution with label 57 may be accessed using the command `data(57)`.

Query string	Meaning
TY name	The short name for the solution type (see Table 4.5).
TY number	The number of the solution type (see Table 4.5).
BR	The branch number.
PT	The point number.
LAB	The solution label, if any.
section	A unique identifier for each branch in a file with multiple branches.
data	An array which contains the AUTO output.

Table 4.4: This table shows the strings that can be used to query a bifurcation diagram object and their meanings.

```

> cp \${AUTO}_DIR/python/demo/userScript.py .
> ls
userScript.py
> cat userScript.py
# This is an example script for the AUTO07p command line user
# interface. See the "Command Line User Interface" chapter in the
# manual for more details.
from AUTOclui import *

def myRun(demo):

    copydemo(demo)

    ld(demo)
    run()
    sv(demo)
    ld(s=demo)
    data = sl(demo)
    ch("NTST",50)
    for solution in data:
        if solution["Type name"] == "BP":
            ch("IRS", solution["Label"])
            ch("ISW", -1)
            # Compute forward
            run()
            ap(demo)
            # Compute back
            ch("DS",-pr("DS"))
            run()
            ap(demo)

    plot(demo)
    wait()

> auto
Initializing
Python 1.5.2 (#1, Feb  1 2000, 16:32:16) [GCC egcs-2.91.66 19990314/Linux
(egcs- on linux-i386
Copyright 1991-1995 Stichting Mathematisch Centrum, Amsterdam
(AUTOInteractiveConsole)
AUTO> from userScript import *
AUTO> myRun('bvp')
...

```

Figure 4.13: This Figure shows the functional version of the AUTO CLUI from Figure 4.12 being used as an extension to the AUTO CLUI. The source code for this script can be found in `\${AUTO}_DIR/python/demo/userScript.py`

```

AUTO> data=dg('ab')
Parsed file: b.ab
AUTO> print data[0]
{'LAB': 6, 'TY name': 'EP', 'data': [0.0, 0.0, 0.0, 0.0], 'section': 12,
'BR': 2, 'PT': 1, 'TY number': 9}
AUTO> print data[0]['LAB']
6
AUTO>

```

Figure 4.14: This figure shows an example of parsing a bifurcation diagram. The first command, `data=dg('ab')`, reads in the bifurcation diagram and puts it into the variable `data`. The second command, `print data[0]` prints out all of the data about the first solution in the list. The third command, `print data[0]['LAB']`, prints out the label of the first point.

Type	Short Name	Number
No Label	No Label	
Branch point (algebraic problem)	BP	1
Fold (algebraic problem)	LP	2
Hopf bifurcation (algebraic problem)	HB	3
Regular point (every NPR steps)	RG	4
User requested point	UZ	-4
Fold (ODE)	LP	5
Bifurcation point (ODE)	BP	6
Period doubling bifurcation (ODE)	PD	7
Bifurcation to invariant torus (ODE)	TR	8
Normal begin or end	EP	9
Abnormal termination	MX	-9

Table 4.5: This table shows the the various types of points that can be in solution and bifurcation diagram files, with their short names and numbers.

4.8 Solution Files

Using the `commandParseSolutionFile` command (Section 4.14.19 in the reference) the user can parse and read into memory an AUTO bifurcation solution file. For example, the command `commandParseSolutionFile('ab')` would parse the file `b.ab` (if you are using the standard filename translations from Table 4.3) and return an object which encapsulates the bifurcation solution in a easy to use form.

The object returned by the `commandParseSolutionFile` is a list of all of the solutions in the appropriate bifurcation solution file, and each solution is a Python dictionary with entries for each piece of data for the solution. For example, the sequence of commands in Figure 4.15, prints out the label of the first solution in a bifurcation solution. The queriable parts of the object are listed in Table 4.6.

```
AUTO> data=sl()
Parsed file: fort.8
AUTO> print data[0]
'Branch number': 2
'ISW': 1
'Label': 6
'NCOL': 0
'NTST': 0
'Parameters': [0.0, 14.0, 2.0, 0.0, 0.0, 0.0]
'Point number': 1
'Type name': 'EP'
'Type number': 9
'p': [0.0, 14.0, 2.0, 0.0, 0.0, 0.0]
'parameters': [0.0, 14.0, 2.0, 0.0, 0.0, 0.0]
AUTO> print data[0]['Label']
6
AUTO> data[0]["data"][0]
{'t': 0.0, 'u': [0.0, 0.0]}
```

Figure 4.15: This figure shows an example of parsing a bifurcation solution. The first command, `data=dg('ab')`, reads in the bifurcation solution and puts it into the variable `data`. The second command, `print data[0]` prints out all of the data about the first solution in the list. The third command, `print data[0]['Label']`, prints out the label of the first point. The last command prints the value of the solution at the first point of the first solution.

The individual elements of the array may be accessed in two ways, either by the index of the solution using the `[]` syntax or by label number using the `()` syntax. For example, assume that the parsed object is contained in a variable `data`. The first solution may be accessed using the command `data[0]`, while the solution with label 57 may be accessed using the command `data(57)`.

Query string	Meaning
data	An array which contains the AUTO output.
Branch number	The number of the branch to which the solution belongs.
ISW	The ISW value used to start the calculation. See Section 10.8.3.
Label	The label of the solution.
NCOL	The number of collocation points used to compute the solution. See Section 10.3.2.
NTST	The number of mesh intervals used to compute the solution. See Section 10.3.1.
Parameters	The value of all of the parameters for the solution.
Point number	The number of the point in the given branch.
Type name	A short string which describes the type of the solution (see Table 4.5).
Type number	A number which describes the type of the solution (see Table 4.5).
p	The value of all of the parameters for the solution. (This is an alias for 'Parameter').
parameters	The value of all of the parameters for the solution. (This is an alias for 'Parameter').

Table 4.6: This table shows the strings that can be used to query a bifurcation solution object and their meanings.

4.9 Exporting output data for use by Python or external visualization tools.

The bifurcation and solution file classes have two methods that are particularly useful for creating data which can be used in other programs. First, there is a method called `toArray` which takes a bifurcation diagram or solution and returns a standard Python array. Second, there is a method called `writeRawFilename` which will create a standard ASCII file which contains the bifurcation diagram or the solution. In the solution ASCII file, the first element of each row will be the 't' value and the following elements will be the values of the components at that 't' value. Such ASCII files can be readily parsed and plotted by external tools such as Gnuplot and MATLAB.

For example, we assume that the parsed bifurcation diagram is contained in a variable `b`, for instance, using `b=dg('ab')`. If one wanted to have the bifurcation diagram returned as a Python array one would type `b.toArray()`. Similarly, if one wanted to write out the bifurcation diagram to the file `outputfile` one would type `b.writeRawFilename('outputfile')`.

Now we assume that the parsed solution file is contained in a variable `s`, for example, using `s=sl('ab')`. To get the solution with label 57 returned as a Python array one would type `s(57).toArray()`. Similarly, if one wanted to write out the solution to the file `outputfile` one would type `s(57).writeRawFilename('outputfile')`.

4.10 The .autorc File

Much of the default behavior of the AUTO CLUI can be controlled by the `.autorc` file. The `.autorc` file can exist in either the main AUTO directory, the users home directory, or the current directory. For any option which is defined in more than one file, the `.autorc` file in the current directory (if it exists) takes precedence, followed by the `.autorc` file in the users home directory (if it exists), and then the `.autorc` file in the main AUTO directory. Hence, options may be defined on either a per directory, per user, or global basis.

The first section of the `.autorc` file begins with the line `[AUTO_command_aliases]` and this section defines short names, or aliases, for the AUTO CLUI commands. Each line thereafter is a definition of a command, similar to `branchPoint =commandQueryBranchPoint`. The right hand side of the assignment is the internal AUTO CLUI name for the command, while the left hand side is the desired alias. Aliases and internal names may be used interchangeably, but the intention is that the aliases will be more commonly used. A default set of aliases is provided, and these aliases will be used in the examples in the rest of this Chapter. The default aliases are listed in the reference in Section 4.14.

NOTE: Defaults for the plotting tool may be included in the .autorc file as well. The documentation for this is under development, but the file `$AUTO_DIR/.autorc` contains examples of how these options may be set.

4.11 Two Dimensional Plotting Tool

The two dimensional plotting tool can be run by using the command `plot()` to plot the files `fort.7` and `fort.8` after a calculation has been run, or using the command `plot('foo')` to plot the data in the files `s.foo` and `b.foo`.

The menu bar provides two buttons. The **File** button brings up a menu which allows the user to save the current plot as a Postscript file or to quit the plotter. The **Options** button allows the plotter configuration options to be modified. The available options are described in Table 4.7. In addition, the options can be set from within the CLUI. For example, the set of commands in Figure 4.16 shows how to create a plotter and change its background color to black. The demo script `auto/07p/demo/python/plotter.py` contains several examples of changing options in plotters.

Pressing the right mouse button in the plotting window brings up a menu of buttons which control several aspects of the plotting window. The top two toggle buttons control what function the left button performs. The **print value** button causes the left button to print out the numerical value underneath the pointer when it is clicked. When **zoom** button is checked the left mouse button may be held down to create a box in the plot. When the left button is released the plot will zoom to the selected portion of the diagram. The **unzoom** button returns the diagram to the default zoom. The **Postscript** button allows the user to save the plot as a Postscript file. The **Configure...** button brings up the dialog for setting configuration options.

```
AUTO> plot=pl()
Created plotter
AUTO> plot.config(bg="black")
AUTO>
```

Figure 4.16: This example shows how a plotter is created, and how the background color may be changed to black. All other configuration options are set similarly. Note, the above commands assume that the files `fort.7` and `fort.8` exist in the current directory.

Query string	Meaning
<code>background</code>	The background color of the plot.
<code>bifurcation_column_defaults</code>	A set of bifurcation columns the user is likely to use.
<code>bifurcation_diagram</code>	A parsed bifurcation diagram file to plot.
<code>bifurcation_diagram_filename</code>	The filename of the bifurcation diagram to plot.
<code>bifurcation_symbol</code>	The symbol to use for bifurcation points.
<code>bifurcation_x</code>	The column to plot along the X-axis for bifurcation diagrams.
<code>bifurcation_y</code>	The column to plot along the Y-axis for bifurcation diagrams.
<code>color_list</code>	A list of colors to use for multiple plots.
<code>decorations</code>	Turn on or off the axis, tick marks, etc.
<code>error_symbol</code>	The symbol to use for error points.
<code>foreground</code>	The background color of the plot.
<code>grid</code>	Turn on or off the grid.

hopf_symbol	The symbol to use for Hopf bifurcation points.
index	An array of indices to plot.
label	An array of labels to plot.
label_defaults	A set of labels that the user is likely to use.
limit_point_symbol	The symbol to use for limit points.
mark_t	The t value to marker with a small ball.
maxx	The upper bound for the x-axis of the plot.
maxy	The upper bound for the y-axis of the plot.
minx	The lower bound for the x-axis of the plot.
miny	The lower bound for the y-axis of the plot.
period_doubling_symbol	The symbol to use for period doubling bifurcation points.
ps_colormode	The PostScript output mode: 'color', 'gray' or 'monochrome'.
runner	The runner object from which to get data.
special_point_colors	An array of colors used to mark special points.
special_point_radius	The radius of the spheres used to mark special points.
solution	A parsed solution file to plot.
solution_column_defaults	A set of solution columns the user is likely to use.
solution_filename	The filename of the solution to plot.
solution_x	The column to plot along the X-axis for solutions.
solution_y	The column to plot along the Y-axis for solutions.
symbol_font	The font to use for marker symbols.
symbol_color	The color to use for the marker symbols.
tick_label_template	A string which defines the format of the tick labels.
tick_length	The length of the tick marks.
torus_symbol	The symbol to use for torus bifurcation points.
type	The type of the plot, either "solution" or "bifurcation".
use_labels	Whether or not to display label numbers in the graph.
user_point_symbol	The symbol to use for user defined output points.
xlabel	The label for the x-axis.
xmargin	The margin between the graph and the right and left edges.
xticks	The number of ticks on the x-axis.
ylabel	The label for the y-axis.
ymargin	The margin between the graph and the top and bottom edges.
yticks	The number of ticks on the y-axis.

Table 4.7: This table shows the options that can be set for the AUTO CLUI two dimensional plotting window and their meanings.

4.12 Three Dimensional Plotting Tool

The AUTO three dimensional plotting tool, PLAUTO4, as described in Chapter 8, can be run from the Python CLUI using the command `plot3` or `commandPlotter3D`, in a similar fashion to `plot`. It does not use the options that are used for the two-dimensional plotting window.

4.13 Quick Reference

In this section we have created a table of all of the AUTO CLUI commands, their abbreviations, and a one line description of what function they perform. Each command may be entered using its full name or any of its aliases.

Command	Aliases	Description
<code>commandAppend</code>	<code>ap append</code>	Append data files.
<code>commandCat</code>	<code>cat</code>	Print the contents of a file
<code>commandCd</code>	<code>cd</code>	Change directories.
<code>commandClean</code>	<code>clean cl</code>	Clean the current directory.
<code>commandCopyAndLoadDemo</code>	<code>dm demo</code>	Copy a demo into the current directory and load it.
<code>commandCopyDataFiles</code>	<code>copy cp</code>	Copy data files.
<code>commandCopyDemo</code>	<code>copydemo</code>	Copy a demo into the current directory.
<code>commandCopyFortFiles</code>	<code>sv save</code>	Save data files.
<code>commandCreateGUI</code>	<code>gui</code>	Show AUTOs graphical user interface.
<code>commandDeleteDataFiles</code>	<code>delete dl</code>	Delete data files.
<code>commandDeleteFortFiles</code>	<code>df deletafort</code>	Clear the current directory of fort files.
<code>commandDouble</code>	<code>double db</code>	Double a solution.
<code>commandInteractiveHelp</code>	<code>man help</code>	Get help on the AUTO commands.
<code>commandLs</code>	<code>ls</code>	List the current directory.
<code>commandMoveFiles</code>	<code>move mv</code>	Move data-files to a new name.
<code>commandParseConstantsFile</code>	<code>cn constantsget</code>	Get the current continuation constants.
<code>commandParseDiagramAndSolutionFile</code>	<code>bt diagramandsolutionget</code>	Parse both bifurcation diagram and solution.
<code>commandParseDiagramFile</code>	<code>dg diagramget</code>	Parse a bifurcation diagram.
<code>commandParseSolutionFile</code>	<code>sl solutionget</code>	Parse solution file:
<code>commandPlotter</code>	<code>p2 pl plot</code>	2D plotting of data.
<code>commandPlotter3D</code>	<code>plot3 p3</code>	3D plotting of data.
<code>commandQueryBranchPoint</code>	<code>br bp branchpoint</code>	Print the “branch-point function”.

commandQueryEigenvalue	eigenvalue ev eg	Print eigenvalues of Jacobian (algebraic case).
commandQueryFloquet	fl floquet	Print the Floquet multipliers.
commandQueryHopf	hb hp hopf lp	Print the value of the “Hopf function”.
commandQueryIterations	iterations it	Print the number of Newton iterations.
commandQueryLimitpoint	lm limitpoint	Print the value of the “limit point function”.
commandQueryNote	nt note	Print notes in info file.
commandQuerySecondaryPeriod	sc secondaryperiod sp	Print value of “secondary-periodic bif. fcn”.
commandQueryStepsize	ss stepsize st	Print continuation step sizes.
commandRun	r run rn	Run AUTO.
commandRunnerConfigFort2	changeconstant cc ch	Modify continuation constants.
commandRunnerConfigFort12	hch	Modify HomCont continuation constants.
commandRunnerLoadName	ld load	Load files into the AUTO runner.
commandRunnerPrintFort2	pc pr printconstant	Print continuation parameters.
commandRunnerPrintFort12	hpr	Print HomCont continuation parameters.
commandShell	shell	Run a shell command.
commandSpecialPointLabels	splabs	Return special labels.
commandTriple	tr triple	Triple a solution.
commandUserData	us userdata	Covert user-supplied data files.
commandWait	wait	Wait for the user to enter a key.

4.14 Reference

4.14.1 commandAppend

Purpose

Append data files.

Description

Type `commandAppend('xxx')` to append the output-files `fort.7`, `fort.8`, `fort.9`, to existing data-files `s.xxx`, `b.xxx`, and `d.xxx` (if you are using the default filename templates). Type `commandAppend('xxx','yyy')` to append existing data-files `s.xxx`, `b.xxx`, and `d.xxx` to data-files `s.yyy`, `b.yyy`, and `d.yyy` (if you are using the default filename templates).

Aliases

`ap append`

4.14.2 commandCat

Purpose

Print the contents of a file

Description

Type `'commandCat xxx'` to list the contents of the file `'xxx'`. This calls the Unix function `'cat'` for reading the file.

Aliases

`cat`

4.14.3 `commandCd`

Purpose

Change directories.

Description

Type '`commandCd xxx`' to change to the directory '`xxx`'. This command understands both shell variables and home directory expansion.

Aliases

`cd`

4.14.4 `commandClean`

Purpose

Clean the current directory.

Description

Type `commandClean()` to clean the current directory. This command will delete all files of the form `fort.*`, `*.o`, and `*.exe`.

Aliases

`clean cl`

4.14.5 `commandCopyAndLoadDemo`

Purpose

Copy a demo into the current directory and load it.

Description

Type `commandCopyAndLoadDemo('xxx')` to copy all files from `auto/07p/demos/xxx` to the current user directory. Here 'xxx' denotes a demo name; e.g., 'abc'. Note that the 'dm' command also copies a Makefile to the current user directory. To avoid the overwriting of existing files, always run demos in a clean work directory. NOTE: This command automatically performs the `commandRunnerLoadName` command as well.

Aliases

dm demo

4.14.6 `commandCopyDataFiles`

Purpose

Copy data files.

Description

Type `commandCopyDataFiles('xxx','yyy')` to copy the data-files `c.xxx`, `d.xxx`, `b.xxx`, and `h.xxx` to `c.yyy`, `d.yyy`, `b.yyy`, and `h.yyy` (if you are using the default filename templates).

Aliases

copy cp

4.14.7 `commandCopyDemo`

Purpose

Copy a demo into the current directory.

Description

Type `commandCopyDemo('xxx')` to copy all files from `auto/07p/demos/xxx` to the current user directory. Here `'xxx'` denotes a demo name; e.g., `'abc'`. Note that the `'dm'` command also copies a Makefile to the current user directory. To avoid the overwriting of existing files, always run demos in a clean work directory.

Aliases

`copydemo`

4.14.8 `commandCopyFortFiles`

Purpose

Save data files.

Description

Type `commandCopyFortFiles('xxx')` to save the output-files `fort.7`, `fort.8`, `fort.9`, to `b.xxx`, `s.xxx`, `d.xxx` (if you are using the default filename templates). Existing files with these names will be overwritten.

Aliases

`sv save`

4.14.9 `commandCreateGUI`

Purpose

Show AUTOs graphical user interface.

Description

Type `commandCreateGUI()` to start AUTOs graphical user interface.

NOTE: This command is not implemented yet.

Aliases

`gui`

4.14.10 `commandDeleteDataFiles`

Purpose

Delete data files.

Description

Type `commandDeleteDataFiles('xxx')` to delete the data-files `d.xxx`, `b.xxx`, and `s.xxx` (if you are using the default filename templates).

Aliases

`delete dl`

4.14.11 `commandDeleteFortFiles`

Purpose

Clear the current directory of fort files.

Description

Type `commandDeleteFortFiles()` to clean the current directory. This command will delete all files of the form `fort.*`.

Aliases

`df deletelfort`

4.14.12 `commandDouble`

Purpose

Double a solution.

Description

Type `commandDouble()` to double the solution in 'fort.7' and 'fort.8'.

Type `commandDouble('xxx')` to double the solution in `b.xxx` and `s.xxx` (if you are using the default filename templates).

Aliases

`double db`

4.14.13 `commandInteractiveHelp`

Purpose

Get help on the AUTO commands.

Description

Type 'help' to list all commands with a online help. Type 'help xxx' to get help for command 'xxx'.

Aliases

man help

4.14.14 `commandLs`

Purpose

List the current directory.

Description

Type 'commandLs' to run the system 'ls' command in the current directory. This command will accept whatever arguments are accepted by the Unix command 'ls'.

Aliases

ls

4.14.15 `commandMoveFiles`

Purpose

Move data-files to a new name.

Description

Type `commandMoveFiles('xxx','yyy')` to move the data-files `b.xxx`, `s.xxx`, `d.xxx`, and `c.xxx` to `b.yyy`, `s.yyy`, `d.yyy`, and `c.yyy` (if you are using the default filename templates).

Aliases

`move mv`

4.14.16 `commandParseConstantsFile`

Purpose

Get the current continuation constants.

Description

Type `commandParseConstantsFile('xxx')` to get a parsed version of the constants file `c.xxx` (if you are using the default filename templates).

Aliases

`cn constantsget`

4.14.17 `commandParseDiagramAndSolutionFile`

Purpose

Parse both bifurcation diagram and solution.

Description

Type `commandParseDiagramAndSolutionFile('xxx')` to get a parsed version of the diagram file `b.xxx` and solution file `s.xxx` (if you are using the default filename templates).

Aliases

`bt diagramandsolutionget`

4.14.18 `commandParseDiagramFile`

Purpose

Parse a bifurcation diagram.

Description

Type `commandParseDiagramFile('xxx')` to get a parsed version of the diagram file `b.xxx` (if you are using the default filename templates).

Aliases

`dg diagramget`

4.14.19 `commandParseSolutionFile`

Purpose

Parse solution file:

Description

Type `commandParseSolutionFile('xxx')` to get a parsed version of the solution file `s.xxx` (if you are using the default filename templates).

Aliases

`sl solutionget`

4.14.20 `commandPlotter`

Purpose

2D plotting of data.

Description

Type `commandPlotter('xxx')` to run the graphics program for the graphical inspection of the data-files `b.xxx` and `s.xxx` (if you are using the default filename templates). The return value will be the handle for the graphics window.

Type `commandPlotter()` to run the graphics program for the graphical inspection of the output-files `'fort.7'` and `'fort.8'`. The return value will be the handle for the graphics window.

Aliases

`p2 pl plot`

4.14.21 `commandPlotter3D`

Purpose

3D plotting of data.

Description

Type `commandPlotter3D('xxx')` to run the graphics program for the graphical inspection of the data-files `b.xxx` and `s.xxx` (if you are using the default filename templates). The return value will be the handle for the graphics window.

Type `commandPlotter3D()` to run the graphics program for the graphical inspection of the output-files `'fort.7'` and `'fort.8'`. The return value will be the handle for the graphics window.

Aliases

`plot3 p3`

4.14.22 `commandQueryBranchPoint`

Purpose

Print the “branch-point function”.

Description

Type `commandQueryBranchPoint()` to list the value of the “branch-point function” in the output-file `fort.9`. This function vanishes at a branch point.

Type `commandQueryBranchPoint('xxx')` to list the value of the “branch-point function” in the info file `'d.xxx'`.

Aliases

`br bp branchpoint`

4.14.23 `commandQueryEigenvalue`

Purpose

Print eigenvalues of Jacobian (algebraic case).

Description

Type `commandQueryEigenvalue()` to list the eigenvalues of the Jacobian in `fort.9`. (Algebraic problems.)

Type `commandQueryEigenvalue('xxx')` to list the eigenvalues of the Jacobian in the info file `'d.xxx'`.

Aliases

`eigenvalue` `ev` `eg`

4.14.24 `commandQueryFloquet`

Purpose

Print the Floquet multipliers.

Description

Type `commandQueryFloquet()` to list the Floquet multipliers in the output-file `fort.9`. (Differential equations.)

Type `commandQueryFloquet('xxx')` to list the Floquet multipliers in the info file `'d.xxx'`.

Aliases

`fl` `floquet`

4.14.25 `commandQueryHopf`

Purpose

Print the value of the “Hopf function”.

Description

Type `commandQueryHopf()` to list the value of the “Hopf function” in the output-file `fort.9`. This function vanishes at a Hopf bifurcation point.

Type `commandQueryHopf('xxx')` to list the value of the “Hopf function” in the info file `'d.xxx'`.

Aliases

`hb hp hopf lp`

4.14.26 `commandQueryIterations`

Purpose

Print the number of Newton iterations.

Description

Type `commandQueryIterations()` to list the number of Newton iterations per continuation step in `fort.9`.

Type `commandQueryIterations('xxx')` to list the number of Newton iterations per continuation step in the info file `'d.xxx'`.

Aliases

`iterations it`

4.14.27 `commandQueryLimitpoint`

Purpose

Print the value of the “limit point function”.

Description

Type `commandQueryLimitpoint()` to list the value of the “limit point function” in the output-file `fort.9`. This function vanishes at a limit point (fold).

Type `commandQueryLimitpoint('xxx')` to list the value of the “limit point function” in the info file `'d.xxx'`.

Aliases

`lm limitpoint`

4.14.28 `commandQueryNote`

Purpose

Print notes in info file.

Description

Type `commandQueryNote()` to show any notes in the output-file `fort.9`.

Type `commandQueryNote('xxx')` to show any notes in the info file `'d.xxx'`.

Aliases

`nt note`

4.14.29 `commandQuerySecondaryPeriod`

Purpose

Print value of “secondary-periodic bif. fcn”.

Description

Type `commandQuerySecondaryPeriod()` to list the value of the “secondary-periodic bifurcation function” in the output-file 'fort.9'. This function vanishes at period-doubling and torus bifurcations.

Type `commandQuerySecondaryPeriod('xxx')` to list the value of the “secondary-periodic bifurcation function” in the info file 'd.xxx'.

Aliases

sc secondaryperiod sp

4.14.30 `commandQueryStepsize`

Purpose

Print continuation step sizes.

Description

Type `commandQueryStepsize()` to list the continuation step size for each continuation step in 'fort.9'.

Type `commandQueryStepsize('xxx')` to list the continuation step size for each continuation step in the info file 'd.xxx'.

Aliases

ss stepsize st

4.14.31 `commandRun`

Purpose

Run AUTO.

Description

Type `commandRun([options])` to run AUTO with the given options. There are four possible options:

Long name	Short name	Description
equation	e	The equations file
constants	c	The AUTO constants file
solution	s	The restart solution file
homcont	h	The Homcont parameter file

Options which are not explicitly set retain their previous value. For example one may type: `commandRun(e='ab',c='ab.1')` to use 'ab.c' as the equations file and c.ab.1 as the constants file (if you are using the default filename templates).

You can also specify an `sv='xxx'` option to save to b.xxx, and so on, or ap to append, or AUTO Constants, e.g., `DS=0.05`, or `IRS=2`. Special values for DS are '+' (forwards) and '-' (backwards).

Type `commandRun('name')` load all files with base 'name'. This does the same thing as running `commandRun(e='name',c='name',s='name',h='name')`.

`run('name','save')` does the same thing as running `run(e='name',c='name',s='name',h='name',sv='save')`.

Aliases

r run rn

4.14.32 `commandRunnerConfigFort2`

Purpose

Modify continuation constants.

Description

Type `commandRunnerConfigFort2('xxx',yyy)` to change the constant 'xxx' to have value yyy.

Aliases

`changeconstant cc ch`

4.14.33 `commandRunnerConfigFort12`

Purpose

Modify HomCont continuation constants.

Description

Type `commandRunnerConfigFort12('xxx',yyy)` to change the HomCont constant 'xxx' to have value yyy.

Aliases

`hch`

4.14.34 `commandRunnerLoadName`

Purpose

Load files into the AUTO runner.

Description

Type `commandRunnerLoadName([options])` to modify AUTO runner. There are four possible options:

Long name	Short name	Description
-----	-----	-----
<code>equation</code>	<code>e</code>	The equations file
<code>constants</code>	<code>c</code>	The AUTO constants file
<code>solution</code>	<code>s</code>	The restart solution file
<code>homcont</code>	<code>h</code>	The Homcont parameter file

Options which are not explicitly set retain their previous value. For example one may type: `commandRunnerLoadName(e='ab',c='ab.1')` to use 'ab.c' as the equations file and c.ab.1 as the constants file (if you are using the default filename templates).

Type `commandRunnerLoadName('name')` load all files with base 'name'. This does the same thing as running `commandRunnerLoadName(e='name',c='name',s='name',h='name')`.

You can also specify AUTO Constants, e.g., `DS=0.05`, or `IRS=2`. Special values for DS are '+' (forwards) and '-' (backwards).

Aliases

ld load

4.14.35 `commandRunnerPrintFort2`

Purpose

Print continuation parameters.

Description

Type `commandRunnerPrintFort2()` to print all the parameters. Type `commandRunnerPrintFort2('xxx')` to return the parameter 'xxx'.

Aliases

pc pr printconstant

4.14.36 `commandRunnerPrintFort12`

Purpose

Print HomCont continuation parameters.

Description

Type `commandRunnerPrintFort12()` to print all the parameters. Type `commandRunnerPrintFort12('xxx')` to return the parameter 'xxx'.

Aliases

hpr

4.14.37 `commandShell`

Purpose

Run a shell command.

Description

Type `'shell xxx'` to run the command `'xxx'` in the Unix shell and display the results in the AUTO command line user interface.

Aliases

`shell`

4.14.38 `commandSpecialPointLabels`

Purpose

Return special labels.

Description

Type `splabs('xxx',typename)` to get a list of labels with the specified `typename`, where `typename` can be one of `'EP'`, `'MX'`, `'BP'`, `'LP'`, `'UZ'`, `'HB'`, `'PD'`, `'TR'`, or `'RG'`.
Or use `splabs(s,typename)` where `s` is a parsed solution from `sl()`.

Aliases

`splabs`

4.14.39 `commandTriple`

Purpose

Triple a solution.

Description

Type `commandTriple()` to triple the solution in 'fort.7' and 'fort.8'.

Type `commandTriple('xxx')` to triple the solution in b.xxx and s.xxx (if you are using the default filename templates).

Aliases

tr triple

4.14.40 `commandUserData`

Purpose

Covert user-supplied data files.

Description

Type `commandUserData('xxx')` to convert a user-supplied data file 'xxx.dat' to AUTO format. The converted file is called 's.dat'. The original file is left unchanged. AUTO automatically sets the period in `PAR(10)`. Other parameter values must be set in 'stpnt'. (When necessary, `PAR(10)` may also be redefined there.) The constants-file file 'c.xxx' must be present, as the AUTO-constants 'NTST' and 'NCOL' are used to define the new mesh. For examples of using the 'userData' command see demos 'lor' and 'pen' (where it has the old name 'fc').

Aliases

us userdata

4.14.41 `commandWait`

Purpose

Wait for the user to enter a key.

Description

Type 'commandWait' to have the AUTO interface wait until the user hits any key (mainly used in scripts).

Aliases

wait

Chapter 5

Running AUTO using Unix Commands.

Apart from the Python commands described in the previous chapter, AUTO can also be run with the GUI described in Chapter 9, or using the Unix commands described below. These Unix commands run both directly in the shell and at the AUTO Python prompt. The AUTO aliases must have been activated; see Section 1.1; and an equations-file `xxx.f` and a corresponding constants-file `c.xxx` (see Section 3) must be in the current user directory.

Do not run AUTO in the directory `auto/07p` or in any of its subdirectories.

5.1 Basic commands.

@r : Type `@r xxx` to run AUTO. Restart data, if needed, are expected in `s.xxx`, and AUTO-constants in `c.xxx`. This is the simplest way to run AUTO.

- Type `@r xxx yyy` to run AUTO with equations-file `xxx.f` and restart data-file `s.yyy`. AUTO-constants must be in `c.xxx`.
- Type `@r xxx yyy zzz` to run AUTO with equations-file `xxx.f`, restart data-file `s.yyy` and constants-file `c.zzz`.

@R : The command `@R xxx` is equivalent to the command `@r xxx` above.

- Type `@R xxx i` to run AUTO with equations-file `xxx.f`, constants-file `c.xxx.i` and, if needed, restart data-file `s.xxx`.
- Type `@R xxx i yyy` to run AUTO with equations-file `xxx.f`, constants-file `c.xxx.i` and restart data-file `s.yyy`.

@sv : Type `@sv xxx` to save the output-files `fort.7`, `fort.8`, `fort.9`, as `b.xxx`, `s.xxx`, `d.xxx`, respectively. Existing files by these names will be deleted.

@ap : Type `@ap xxx` to append the output-files `fort.7`, `fort.8`, `fort.9`, to existing data-files `b.xxx`, `s.xxx`, `d.xxx`, resp.

- Type `@ap xxx yyy` to append `b.xxx`, `s.xxx`, `d.xxx`, to `b.yyy`, `s.yyy`, `d.yyy`, resp.

5.2 Plotting commands.

- @p** : Type *@p xxx* to run the graphics program PLAUT (See Chapter 7) for the graphical inspection of the data-files **b.xxx** and **s.xxx**.
- Type *@p* to run the graphics program PLAUT for the graphical inspection of the output-files **fort.7** and **fort.8**.
- @ps** : Type *@ps fig.x* to convert a saved PLAUT figure **fig.x** from compact PLOT10 format to PostScript format. The converted file is called **fig.x.ps**. The original file is left unchanged.

5.3 File-manipulation.

- @cp** : Type *@cp xxx yyy* to copy the data-files **b.xxx**, **s.xxx**, **d.xxx**, **c.xxx** to **b.yyy**, **s.yyy**, **d.yyy**, **c.yyy**, respectively.
- @mv** : Type *@mv xxx yyy* to move the data-files **b.xxx**, **s.xxx**, **d.xxx**, **c.xxx**, to **b.yyy**, **s.yyy**, **d.yyy**, **c.yyy**, respectively.
- @df** : Type *@df* to delete the output-files **fort.7**, **fort.8**, **fort.9**.
- @cl** : Type *@cl* to clean the current directory. This command will delete all files of the form **fort.***, ***.o**, and ***.exe**.
- @dl** : Type *@dl xxx* to delete the data-files **b.xxx**, **s.xxx**, **d.xxx**.

5.4 Diagnostics.

- @lp** : Type *@lp* to list the value of the “limit point function” in the output-file **fort.9**. This function vanishes at a limit point (fold).
- Type *@lp xxx* to list the value of the “limit point function” in the data-file **d.xxx**. This function vanishes at a limit point (fold).
- @bp** : Type *@bp* to list the value of the “branch-point function” in the output-file **fort.9**. This function vanishes at a branch point.
- Type *@bp xxx* to list the value of the “branch-point function” in the data-file **d.xxx**. This function vanishes at a branch point.
- @hb** : Type *@hb* to list the value of the “Hopf function” in the output-file **fort.9**. This function vanishes at a Hopf bifurcation point.
- Type *@hb xxx* to list the value of the “Hopf function” in the data-file **d.xxx**. This function vanishes at a Hopf bifurcation point.

- @sp** : Type *@sp* to list the value of the “secondary-periodic bifurcation function” in the output-file **fort.9**. This function vanishes at period-doubling and torus bifurcations.
- Type *@sp xxx* to list the value of the “secondary-periodic bifurcation function” in the data-file **d.xxx**. This function vanishes at period-doubling and torus bifurcations.
- @it** : Type *@it* to list the number of Newton iterations per continuation step in **fort.9**.
- Type *@it xxx* to list the number of Newton iterations per continuation step in **d.xxx**.
- @st** : Type *@st* to list the number of stable eigenvalues or stable Floquet multipliers per continuation step in **fort.9**.
- @ss** : Type *@st* to list the continuation step size for each continuation step in **fort.9**.
- Type *@st xxx* to list the continuation step size for each continuation step in **d.xxx**.
- @ev** : Type *@ev* to list the eigenvalues of the Jacobian in **fort.9**. (Algebraic problems.)
- Type *@ev xxx* to list the eigenvalues of the Jacobian in **d.xxx**. (Algebraic problems.)
- @fl** : Type *@fl* to list the Floquet multipliers in the output-file **fort.9**. (Differential equations.)
- Type *@fl xxx* to list the Floquet multipliers in the data-file **d.xxx**. (Differential equations.)

5.5 File-editing.

- @e7** : To use the vi editor to edit the output-file **fort.7**.
- @e8** : To use the vi editor to edit the output-file **fort.8**.
- @e9** : To use the vi editor to edit the output-file **fort.9**.
- @j7** : To use the SGI jot editor to edit the output-file **fort.7**.
- @j8** : To use the SGI jot editor to edit the output-file **fort.8**.
- @j9** : To use the SGI jot editor to edit the output-file **fort.9**.

5.6 File-maintenance.

- @lb** : Type *@lb* to run an interactive utility program for listing, deleting and relabeling solutions in the output-files **fort.7** and **fort.8**. The original files are backed up as **~fort.7** and **~fort.8**.
- Type *@lb xxx* to list, delete and relabel solutions in the data-files **b.xxx** and **s.xxx**. The original files are backed up as **~b.xxx** and **~s.xxx**.

- Type `@lb xxx yyy` to list, delete and relabel solutions in the data-files `b.xxx` and `s.xxx`. The modified files are written as `b.yyy` and `s.yyy`.

@fc : Type `@fc xxx` to convert a user-supplied data file `xxx.dat` to AUTO format. The converted file is called `s.dat`. The original file is left unchanged. AUTO automatically sets the period in `PAR(11)`. Other parameter values must be set in `STPNT`. (When necessary, `PAR(11)` may also be redefined there.) The constants-file file `c.xxx` must be present, as the AUTO-constants `NTST` and `NCOL` (Sections 10.3.1 and 10.3.2) are used to define the new mesh. For examples of using the `@fc` command see demos `lor` and `pen`.

5.7 HomCont commands.

@h : Use `@h` instead of `@r` when using HomCont, i.e., when `IPS=9` (see Chapter 20). Type `@h xxx` to run AUTO/HomCont. Restart data, if needed, are expected in `s.xxx`, AUTO-constants in `c.xxx` and HomCont-constants in `s.xxx`.

- Type `@h xxx yyy` to run AUTO/HomCont with equations-file `xxx.f` and restart data-file `s.yyy`. AUTO-constants must be in `c.xxx` and HomCont-constants in `s.xxx`.
- Type `@h xxx yyy zzz` to run AUTO/HomCont with equations-file `xxx.f`, restart data-file `s.yyy` and constants-files `c.zzz` and `s.zzz`.

@H : The command `@H xxx` is equivalent to the command `@h xxx` above.

- Type `@H xxx i` in order to run AUTO/HomCont with equations-file `xxx.f` and constants-files `c.xxx.i` and `s.xxx.i` and, if needed, restart data-file `s.xxx`.
- Type `@H xxx i yyy` to run AUTO/HomCont with equations-file `xxx.f`, constants-files `c.xxx.i` and `s.xxx.i`, and restart data-file `s.yyy`.

5.8 Copying a demo.

@dm : Type `@dm xxx` to copy all files from `auto/07p/demos/xxx` to the current user directory. Here `xxx` denotes a demo name; e.g., `abc`. Note that the `@dm` command also copies a *Makefile* to the current user directory. To avoid the overwriting of existing files, always run demos in a clean work directory.

5.9 Viewing the manual.

@mn : Use `gv` to view the PDF version of this manual.

Chapter 6

Output Files.

AUTO writes to standard output and three output-files :

- standard output: A summary of the computation is written to standard output, which usually corresponds to the window in which AUTO is run. Only special, labeled solution points are noted, namely those listed in Table 6.1. The letter codes in the Table are used in the screen output. The numerical codes are used internally and in the `fort.7` and `fort.8` output-files described below.

BP	(1)	Branch point (algebraic systems)
LP	(2)	Fold (algebraic systems)
HB	(3)	Hopf bifurcation
	(4)	User-specified regular output point
UZ	(-4)	Output at user-specified parameter value
LP	(5)	Fold (differential equations)
BP	(6)	Branch point (differential equations)
PD	(7)	Period doubling bifurcation
TR	(8)	Torus bifurcation
EP	(9)	End point of family; normal termination
MX	(-9)	Abnormal termination; no convergence

Table 6.1: Solution Types.

- `fort.7` : The `fort.7` output-file contains the bifurcation diagram. Its format is the same as the `fort.6` (screen) output, but the `fort.7` output is more extensive, as every solution point has an output line printed.
- `fort.8` : The `fort.8` output-file contains complete graphics and restart data for selected, labeled solutions. The information per solution is generally much more extensive than that in `fort.7`. The `fort.8` output should normally be kept to a minimum.
- `fort.9` : Diagnostic messages, convergence history, eigenvalues, and Floquet multipliers are written in `fort.9`. It is strongly recommended that this output be habitually in-

spected. The amount of diagnostic data can be controlled via the AUTO-constant `IID`; see Section 10.9.2.

The user has some control over the `fort.6` (screen) and `fort.7` output via the AUTO-constant `IPLT` (Section 10.9.3). Furthermore, the routine `PVLS` can be used to define “solution measures” which can then be printed by “parameter overspecification”; see Section 10.7.10. For an example see demo `pvl`.

The AUTO-commands `@sv(sv)`, `@ap(ap)`, and `@df(df)` can be used to manipulate the output-files `fort.7`, `fort.8`, and `fort.9`. Furthermore, the AUTO-command `@lb(r1)` can be used to delete and relabel solutions simultaneously in `fort.7` and `fort.8`. For details see Section 5.

The graphics programs `PLAUT`, `PLAUT04`, and the Python CLUI command `plot` can be used to graphically inspect the data in `fort.7` and `fort.8`; see Chapters 7, 8, and 4.

Chapter 7

The Graphics Program PLAUT.

PLAUT can be used to extract graphical information from the AUTO output-files `fort.7` and `fort.8`, or from the corresponding data-files `b.xxx` and `s.xxx`. To invoke PLAUT, use the `@p` command defined in Section 5. The PLAUT window (a Tektronix window) will appear, in which PLAUT commands can be entered. For examples of using PLAUT see the tutorial demo `ab`, in particular, Sections ?? and ??. See also demo `pp2` in Section 14.3.

7.1 Basic PLAUT-Commands.

The principal PLAUT-commands are

- `bd0` : This command is useful for an initial overview of the bifurcation diagram as stored in `fort.7`. If you have not previously selected one of the default options `d0`, `d1`, `d2`, `d3`, or `d4` described below then you will be asked whether you want solution labels, grid lines, titles, or labeled axes.
- `bd` : This command is the same as the `bd0` command, except that you will be asked to enter the minimum and the maximum of the horizontal and vertical axes. This is useful for blowing up portions of a previously displayed bifurcation diagram.
- `ax` : With the `ax` command you can select any pair of columns of real numbers from `fort.7` as horizontal and vertical axis in the bifurcation diagram. (The default is columns 1 and 2). To determine what these columns represent, one can look at the screen output of the corresponding AUTO run, or one can inspect the column headings in `fort.7`.
- `2d` : Upon entering the `2d` command, the labels of all solutions stored in `fort.8` will be listed and you can select one or more of these for display. The number of solution components is also listed and you will be prompted to select two of these as horizontal and vertical axis in the display. Note that the first component is typically the independent time or space variable scaled to the interval $[0,1]$.
- `sav` : To save the displayed plot in a file. You will be asked to enter a file name. Each plot must be stored in a separate new file. The plot is stored in compact PLOT10 format, which can be converted to PostScript format with the AUTO-commands `@ps`; see Section 7.4.

- `cl` : To clear the graphics window.
- `lab` : To list the labels of all solutions stored in `fort.8`. Note that PLAUT requires all labels to be distinct. In case of multiple labels you can use the AUTO command `@lb` to relabel solutions in `fort.7` and `fort.8`.
- `end` : To end execution of PLAUT.

7.2 Default Options.

After entering the commands `bd0`, `bd`, or `2d`, you will be asked whether you want solution labels, grid lines, titles, or axes labels. For quick plotting it is convenient to bypass these selections. This can be done by the default commands `d0`, `d1`, `d2`, `d3`, or `d4` below. These can be entered as a single command or they can be entered as prefixes in the `bd0` and `bd` commands. Thus, for example, one can enter the command `d1bd0`.

- `d0` : Use solid curves, showing solution labels and symbols.
- `d1` : Use solid curves, except use dashed curves for unstable solutions and for solutions of unknown stability. Show solution labels and symbols.
- `d2` : As `d1`, but with grid lines.
- `d3` : As `d1`, except for periodic solutions use solid circles if stable, and open circles if unstable or if the stability is unknown.
- `d4` : Use solid curves, without labels and symbols.

If no default option `d0`, `d1`, `d2`, `d3`, or `d4` has been selected or if you want to override a default feature, then the the following commands can be used. These can be entered as individual commands or as prefixes. For example, one can enter the command `sydpbd0`.

- `sy` : Use symbols for special solution points, for example, open square = branch point, solid square = Hopf bifurcation.
- `dp` : “Differential Plot”, i.e., show stability of the solutions. Solid curves represent stable solutions. Dashed curves are used for unstable solutions and for solutions of unknown stability. For periodic solutions use solid/open circles to indicate stability/instability (or unknown stability).
- `st` : Set up titles and axes labels.
- `nu` : Normal usage (reset special options).

7.3 Other PLAUT-Commands.

The full PLAUT program has several other capabilities, for example,

`scr` : To change the diagram size.

`rss` : To change the size of special solution point symbols.

7.4 Printing PLAUT Files.

`@ps` : Type *@ps fig.1* to convert a saved PLAUT file `fig.1` to PostScript format in `fig.1.ps`.

Chapter 8

The Graphics Program PLAUT04.

“PLAUT04” is a graphic tool for AUTO data visualization. Here we explain how to view AUTO data sets with PLAUT04. An AUTO data set contains a solution file, “s.foo”, a bifurcation file, “b.foo”, and a diagnostic file, “d.foo”. Here “foo” denotes a user-chosen data set name. This user’s guide includes the following information:

1. A description of the PLAUT04 window system.
2. A list of PLAUT04 configuration options.
3. An example of using PLAUT04.

8.1 Quick start

8.1.1 Starting and stopping Plaut04

Starting

The starting command for PLAUT04 is: “plaut04”. A short Unix command is also provided as “@pl”. In the Python CLUI, one can start PLAUT04 by typing “plot3()”, “p3()”, or “commandPlotter3D()”.

This command can have no argument, one argument, or two arguments.

If no argument is provided, then the system uses the AUTO default data files, fort.7, fort.8, and fort.9, as inputs.

If one argument is given, it must be the name of the data set which we want to view. This data set should be in the current directory.

When two arguments are given, the first is always the path to the data set, and the second is the data set name.

Note that the AUTO data set name does not mean the full name of an AUTO file. It refers to the postfix of AUTO data files. For example, if we have the AUTO data files: “s.H1”, “b.H1”, and “d.H1”, the AUTO data file name is “H1”.

Stopping

One can exit the system by clicking the cross at the top-right corner of the window or from the “File” menu of the system.

8.1.2 Changing the “Type”

Often one will frequently change between the solution diagram and the bifurcation diagram. The “Type” menu helps to complete this change. This menu includes two items, “Solution”, and “Bifurcation”. There is a marker beside the current diagram. For example, if the current diagram is the solution diagram, but we want to change to the bifurcation diagram, we can do so by clicking “Type \rightarrow Bifurcation” to switch to the bifurcation diagram.

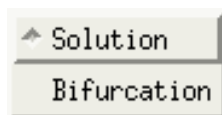


Figure 8.1: The Type Menu

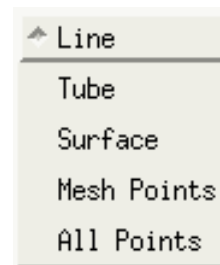


Figure 8.2: The Style Menu

8.1.3 Changing the “Style”

PLAUT04 provides four ways to draw the graphics, *i.e.*, using curves, tubes, points, or as a surface. One can select the style from the “Style” menu. The “Style” menu is shown in Figure 8.2.

8.1.4 Coordinate axes

Figure 8.3 shows the selections of the “Coord” menu. One may use this menu to select to show or not to show coordinate axes, and the type of coordinate axes, in the graphics.

8.1.5 Options

The “Options” Menu provides functions to add or remove widgets from the graphics. It also allows to start/stop solution or orbit animation. The “normalize data” normalizes the raw data to $[0,1]$. “Preference” lets us set preferences for the GUI (see Figure 8.4).

8.1.6 CR3BP animation

The “Center” Menu allows to animate the motion of the three bodies in different coordinate systems. We can animate the motion in a large-primary-centered inertial coordinate system, or in a small-primary-centered inertial system, or in the bary-centered inertial system. Figure 8.5 displays the layout of the “Center” menu.

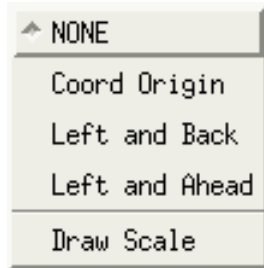


Figure 8.3: The Draw-Coordinate-Axes Menu

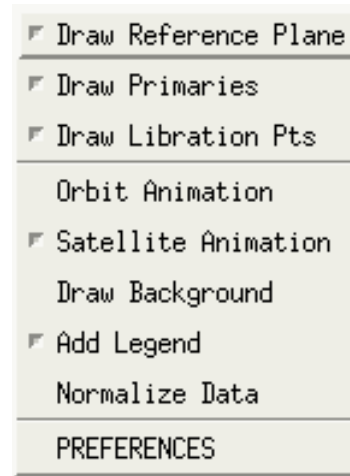


Figure 8.4: The Options Menu

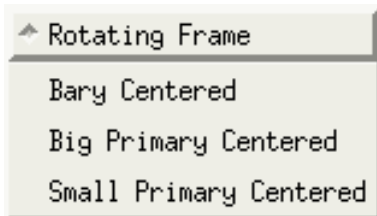


Figure 8.5: The Center Menu

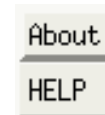


Figure 8.6: The Help Menu

8.1.7 Help

The “Help” menu provides an on-line help on how to use PLAUT04.

8.1.8 Picking a point in the diagram

The picking operation is useful when we want to know data corresponding to a certain point in the diagram. In order to execute a picking operation, we should follow these steps.

- Click the arrow icon to change the mouse to picking state.
- Move the mouse to the point of interest.
- Click the left button of the mouse to pick the point.

Once a point has been picked, a new window is popped up. In this new window, the Floquet multipliers of the point are shown in an x-y plane. Black crosses in the diagram indicate the Floquet Multipliers. The solution, and the values of the corresponding Floquet Multipliers, are given in the lower part of the window. A unit circle is drawn in the diagram. Figure 8.7 is an example of the picking operation. From this diagram, we can see that two Floquet Multipliers are outside the unit circle, two are on the unit circle, and the other two are inside the unit circle.

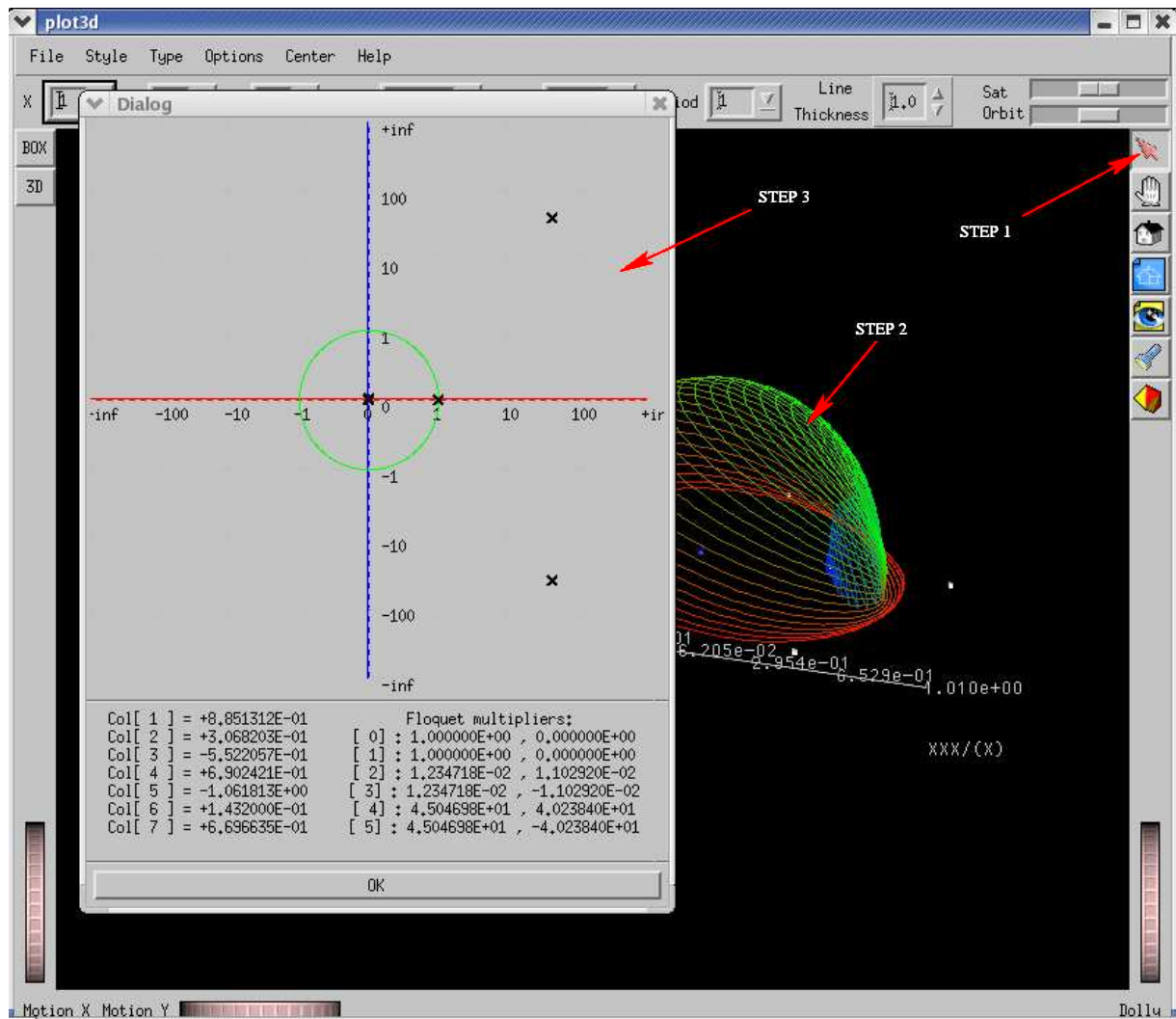


Figure 8.7: Picking a point

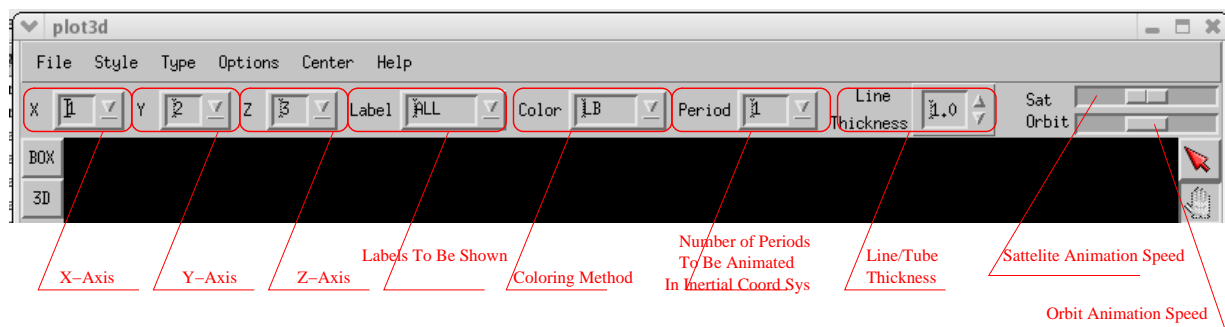


Figure 8.8: Menu-bar layout

8.1.9 Choosing the variables

AUTO can generate large amounts of data. The CR3BP, for example, has 6 variables, *i.e.*, x, y, z, x', y', z' , and time. One can choose to draw any combination of these variables in 2 or 3

dimensions using PLAUT04. On the list bar, we can see three dropdown lists with label “X”, “Y”, and “Z” (See Figure 8.8). Each of these three lists has the exact number of choices, namely, the number of variables of the system plus one. In our case, these lists have 7 choices, which are represented by the integers 0 to 6. 0 represents time. 1 to 6 stand for $x, y, z, x', y',$ and z' , respectively. “1” is selected for “X”, which indicates that x is drawn on the X-axis. “2” is selected for “Y”, which indicates that y is represented on the Y-axis. “3” is selected for “Z”, which indicates that z is represented on the Z-axis.

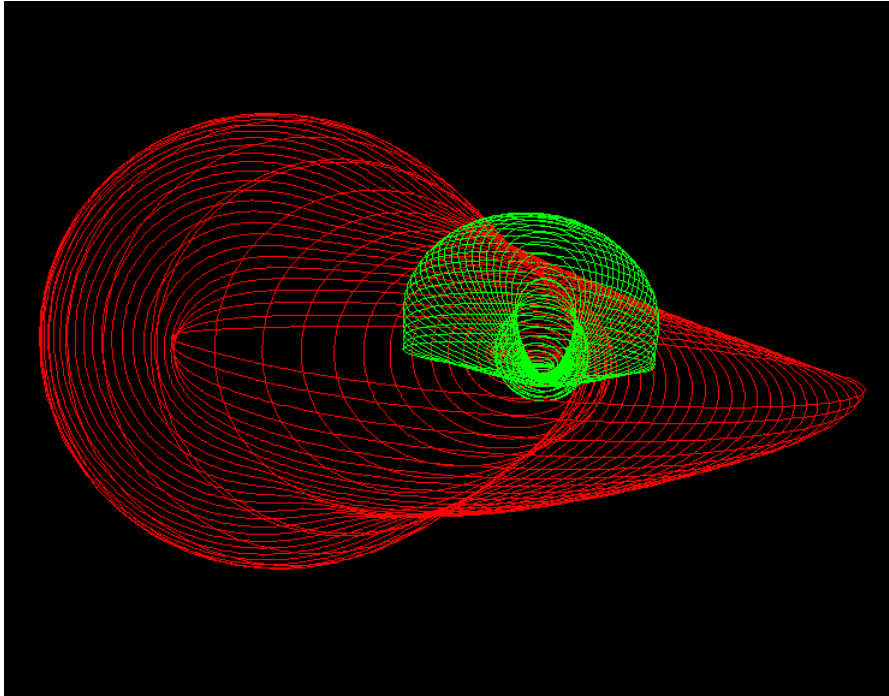


Figure 8.9: Displaying multiple components

We can also show multiple combinations at the same time. For example, if we want to show x - y - z and x' - y' - z' in the same diagram, we can input 1, 4 in the “X” dropdown list to select x and x' being drawn on the X-axis, input 2, 5 in the “Y” list to show y and y' on the Y-axis, and input 3, 6 in the “Z” dropdown list to draw z and z' on the Z-Axis. Note that after finishing the input in the dropdown list box, we must type “ENTER” for the input to be accepted by the system. Figure 8.9 shows the results of the above choices. The combination is flexible. For example, if X is 1, Y is 3, 5, and Z is 4, 5, 6, the system will automatically reorganize them to 1 – 3 – 4, 1 – 5 – 5, 1 – 3 – 6 and show the results. If X is 1, 5, Y is 2, and Z is 3, 4, the system reorganizes them to 1 – 2 – 3, 5 – 2 – 4.

Different components are drawn with different colors from blue to red.

The default values can be set in the resource file. If no resource file exists, then the system will use “1” for X-axis, “2” for Y-axis, and “3” for Z-axis for both the solution and the bifurcation diagrams.

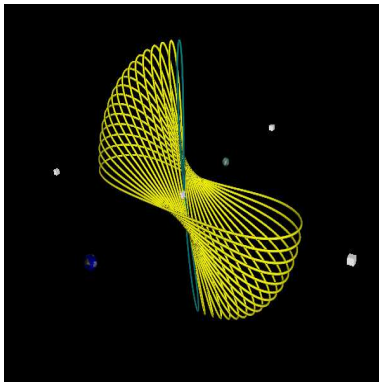
8.1.10 Choosing labels

From the Label list, we can choose the label of the solution to be drawn. If “ALL” is chosen, all solutions are shown in the diagram. If “NONE” is chosen, none of the solutions is shown. “HALF” shows the solutions with odd labels and special solutions only. “SPEC” lets the system show the special solutions only. We can also show selected solutions by inputting their labels in the list box separated by commas. For example, typing 1, 10, 15, 20 will lead the system to show only the solutions with label 1, 10, 15 and 20.

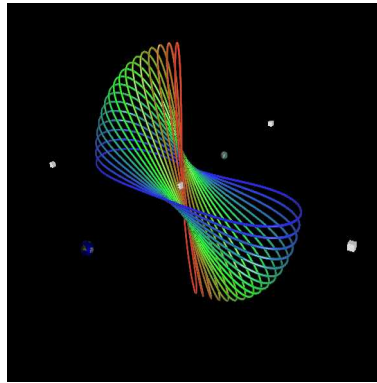
We can set the default value for this list in the PLAUT04 resource file.

8.1.11 Coloring

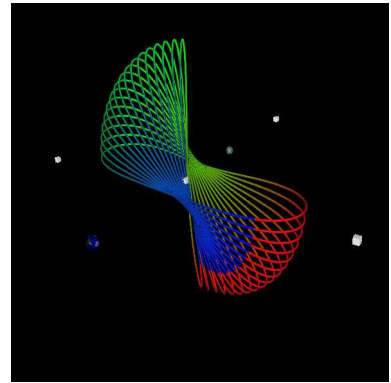
Many coloring methods are provided. They can be classified into three groups. The first group is coloring by variables. This group provides as many choices as the number of variables of a problem plus 1 for the time. The second group is coloring by parameters. These parameters are defined by the AUTO user. in the AUTO constants file. There are as many choices as the number of parameters defined in the AUTO constants file. The third group includes “TYPE”, type of solution, “PONT”, point number, “BRAN”, the branch to which the solution belongs, and “LABL”, label of the solution. Different coloring methods cannot be used at the same time. Figure 8.10 shows the difference between coloring by type and coloring by label. From Figure 8.10(a), we can see that there is only one branching orbit in this family, which is shown in cyan. In Figure 8.10(b), the start solution is colored in blue, and the last solution is colored in red. When using time to color the diagram, 0 is set to blue, while 1 is set to red.



(a) Coloring by “Type”



(b) Coloring by “Label”



(c) Coloring by “Time”

Figure 8.10: Coloring

We can set the default value in the PLAUT04 resource file.

8.1.12 Number of periods to be animated

Generally only one period is animated when we animate the solution in the inertial frame. However, the SpinBox allows us to change the default value. This is a specially designed function for the CR3BP. It is useful when we animate the motion in the three bodies in the inertial frame.

8.1.13 Changing the line/tube thickness

The “Line Thickness” spinbox allows us to increase or decrease the line/tube thickness in the diagram. The PLAUT04 resource file also provides a way to change the default values of the line/tube thickness.

8.1.14 Changing the animation speed

The “Sat” and “Orbit” scale bar allow us to change the animation speed. Their Maximum and Minimum value can be set in the resource file.

8.1.15 Changing the background picture

A user can set the background with his favorite picture. To do this, a user should copy the picture to the directory “\$AUTO_DIR/plaut04/widgets”, and then change the name of the file to “background.rgb”.

8.2 Setting up the resource file

The PLAUT04 resource file sets default values for almost all controls of PLAUT04. PLAUT04 allows us to write our own resources files and put them in the same directory as the AUTO data files. PLAUT04 first looks for the resource file in the current directory. If it cannot find a resource file there, then it will try to use the one installed in the AUTO root directory. If both these searches fail, then the internal default values will be used.

In order to write a usable resource file, one should follow the following rules:

1. Comment lines start with “#”. Comments may take as many lines as desired.
2. Between the “variable name” and the default value, we must use “=” to tell the system that the left side is the “variable name”, and the right side is its corresponding default value.
3. If a “variable” has aggregate values, a comma “,” must be used between two values.
4. The line type is set using 4-digit hexadecimal, starting with “0x”. Its values can range from 0 (invisible) to “0xffff” (solid). The system default is “0xffff” for stable solutions, and “0x3333” for unstable ones. The line pattern is determined by the number of 1s and 0s when the hexadecimal is converted to a 16-bit binary. A “1” indicates that the drawing occurs, and “0” that it does not, on a pixel by pixel basis. For example, the pattern “0xAAAA”, in binary is 0000100010001000, and PLAUT04 interprets this as drawing 3 bits off, 1 bit on, 3 bits off, 1 bit on, 3 bits off, 1 bit on and finally 4 bits off. The pattern is read backward because the low order bits are used first.
5. Some variables can only be set to “Yes” or “No”. They cannot be assigned other values.
6. No “variable name” should be modified.

It is strongly recommended that the default resource file is used as a template when writing a custom resource file.

Below is a copy of the default resource file.

```
#version 0.0

# Line colors are represented by RGB values from 0 to 1.0.
# DEFAULT color is also used when animationLabel == 0, i.e.,
# when showing all solutions and animating the solution change.
# Point Type    RED    GREEN    BLUE    PATTERN
DEFAULT        = 1.0,   1.0,   1.0, 0xffff
BP             = 1.0,   0.0,   0.0, 0xffff
LP ALG        = 0.0,   1.0,   0.0, 0xffff
HB            = 0.0,   0.0,   1.0, 0xffff
UZ4           = 1.0,   1.0,   0.0, 0xffff
UZ-4          = 0.5,   0.5,   0.0, 0xffff
LP DIF        = 0.0,   0.0,   0.5, 0xffff
BP DIF        = 0.0,   0.5,   0.5, 0xffff
PD            = 1.0,   0.0,   1.0, 0xffff
TR            = 0.0,   1.0,   1.0, 0xffff
EP            = 0.3,   0.0,   0.3, 0xffff
MX            = 0.6,   0.0,   0.6, 0xffff
OTHERS        = 1.0,   1.0,   1.0, 0xffff

# Initialize the line pattern for showing stability
UNSTABLE LINE PATTERN = 0xffff
STABLE LINE PATTERN   = 0xffff

# Initialize the default options:
Draw Reference Plane = No
Orbit Animation      = No
Satellite Animation = No
Draw Primaries       = No
Draw Libration Points = No
Normalize Data       = Yes
Draw Background      = No

# Initialize the default coordinate axes:
# 0 --- None,
# 1 --- at origin
# 2 --- at left and behind
# 3 --- at left and ahead
Coordinate Type = 3

# Draw Scale on the Aexs
Draw Scale = Yes

# Initialize the default graph type:
# 0 --- Solution (fort.8)
# 1 --- Bifurcation (fort.7)
Graph Type = 0

# Initialize the default graph style:
# 0 --- LINES,
```

```

# 1 --- TUBES,
# 2 --- SURFACE
Graph Style = 0

# Set the window width and height:
Window Width      = 1000
Window Height     = 1000

# Set X, Y, Z axes for the solution diagram:
# 0 is Time for X,Y,Z.
X Axis Solution   = 1
Y Axis Solution   = 2
Z Axis Solution   = 3

# Set X, Y, Z axes for the bifurcation diagram:
X Axis Bifurcation = 4
Y Axis Bifurcation = 5
Z Axis Bifurcation = 6

#Labeled solutions:
Labels           = 0

# Set coloring method:
# -5 --- STABILITY
# -4 --- POINT
# -3 --- BRANCH
# -2 --- TYPE
# -1 --- LABEL
# Otherwise, according to the data in the ith column of the solution file.
# It can only be set to an integer value.
Coloring Method   = -2
Number of Period Animated = 1

# Line Width Scaler adjusts the thickness of curves:
Line Width Scaler = 1.0

# The AniLine Thickness Scaler sets the thickness of animated solution curves:
AniLine Thickness Scaler = 3.0

# Background color:
Background Color = 0.0, 0.0, 0.0

# Background transparency:
Background Transparency = 0.0

# Disk transparency
# IF you turn Disk From File to "Yes", you should change the transparency there.
Disk Transparency = 0.7

# Read Disk From File
Disk From File = No

# Axes color:
X Axis Color = 1.0, 0.0, 0.0

```

```

Y Axis Color      = 0.0,  1.0,   0.0
Z Axis Color      = 0.0,  0.0,   1.0

# Color of the satellite, large primary, and small primary in animation:
satellite Color   = 1.0,  0.0,   0.0
large primary Color = 0.0,  1.0,   0.0
large primary tail Color = 0.0,  1.0,   1.0
small primary Color = 0.0,  0.0,   1.0
small primary tail Color = 0.5,  0.5,   0.0

# Stable solution color:
Stable Solution Color = 0.0, 0.0, 1.0

# Stable solution color:
Unstable Solution Color = 1.0, 0.0, 0.0

# Set the radius of the satellite, large primary, and small primary:
# The normal size is 1.0.
# For smaller radius, use 0.xxx
# For bigger radius, use X.XXX
Satellite Radius   = 1.0
Large Primary Radius = 1.0
Small Primary Radius = 1.0
Libration Point Size = 1.0

# Set the maximum and minimum satellite animation speed:
Sat Max Animation Speed = 100
Sat Min Animation Speed = 0

# Set the maximum and minimum orbit-change animation speed:
Orbit Max Animation Speed = 100
Orbit Min Animation Speed = 0

# Set the active AUTO parameter indices:
parameter ID = 10

# Choose 3D or 2D graph:
3D = Yes

```

8.3 Example

In this example, we want to view a CR3BP data set. We want the diagram to show the “ x ” component on the X-axis, “ y ” component on the Y-axis, and “ z ” component on the Z-axis for the solution diagram. In the CR3BP, we use the parameters “1 2 3 10 21 22 23” in the AUTO calculations, and we also want to be able to use these to color the diagram, so we set the “parameter indices”.

Other preferences include

- The diagram is drawn using Tubes.
- Coordinate axes are not drawn.

- No animation.
- Reference plane, libration points, and primaries are drawn.
- All labels are shown.
- Data is not normalized.

The settings are the settings in the resource file are then as follows:

```
# Initialize the default options
Draw Reference Plane = Yes
Orbit Animation      = No
Satellite Animation  = No
Draw Primaries       = Yes
Draw Libration Points = Yes
Normalize Data        = No
Draw Background      = No

# Initialize the default graph type
# 0 --- Solution(fort.8) 1 --- Bifurcation(fort.7)
Graph Type           = 0

# initialize the default graph style
# 0 --- LINES, 1 --- TUBES, 2 ---- SURFACE 3--- nurbs curve
graph Style          = 1

# set X, Y, Z, and Label
# 0 is Time for X,Y,Z. 0 is "All" for Label

Solution X Axis      = 1
Solution Y Axis      = 2
Solution Z Axis      = 3

Labels               = 0

#set the parameter indices
parameter ID = 1, 2, 3, 10, 15, 21, 22, 23
```

Based on the above settings, the solution diagram for the CR3BP family L1 for $\mu = 0.01215$ appears in Figure 8.11.

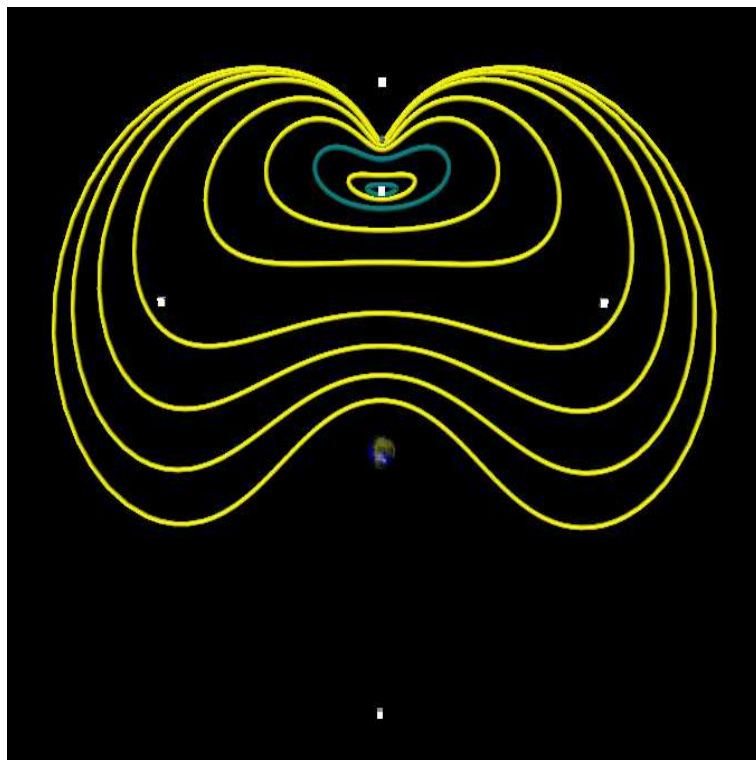


Figure 8.11: Example

Chapter 9

The Graphical User Interface GUI94.

9.1 General Overview.

The AUTO graphical user interface (GUI) is a tool for creating and editing equations-files and constants-files; see Section 3 for a description of these files. The GUI can also be used to run AUTO and to manipulate and plot output-files and data-files; see Section 5 for corresponding commands. To use the GUI for a new equation, change to an empty work directory. For an existing equations-file, change to its directory. (*Do not activate the GUI in the directory auto/07p or in any of its subdirectories.*) Then type

@auto,

or its abbreviation @a. Here we assume that the AUTO aliases have been activated; see Section 1.1. The GUI includes a window for editing the equations-file, and four groups of buttons, namely, the *Menu Bar* at the top of the GUI, the *Define Constants*-buttons at the center-left, the *Load Constants*-buttons at the lower left, and the *Stop- and Exit*-buttons.

Note : Most GUI buttons are activated by point-and-click action with the *left* mouse button. If a beep sound results then the *right* mouse button must be used.

9.1.1 The Menu bar.

It contains the main buttons for running AUTO and for manipulating the equations-file, the constants-file, the output-files, and the data-files. In a typical application, these buttons are used from left to right. First the *Equations* are defined and, if necessary, *Edited*, before being *Written*. Then the AUTO-constants are *Defined*. This is followed by the actual *Run* of AUTO. The resulting output-files can be *Saved* as data-files, or they can be *Appended* to existing data-files. Data-files can be *Plotted* with the graphics program PLAUT, and various file operations can be done with the *Files*-button. Auxiliary functions are provided by the *Demos*-, *Misc*-, and *Help*-buttons. The Menu Bar buttons are described in more detail in Section 9.2.

9.1.2 The Define-Constants-buttons.

These have the same function as the *Define*-button on the Menu Bar, namely to set and change AUTO-constants. However, for the *Define*-button all constants appear in one panel, while for

the Define Constants-buttons they are grouped by function, as in Chapter 10, namely *Problem* definition constants, *Discretization* constants, convergence *Tolerances*, continuation *Step Size*, diagram *Limits*, designation of free *Parameters*, constants defining the *Computation*, and constants that specify *Output* options.

9.1.3 The Load-Constants-buttons.

The *Previous*-button can be used to load an existing AUTO-constants file. Such a file is also loaded, if it exists, by the *Equations*-button on the *Menu Bar*. The *Default*-button can be used to load default values of all AUTO-constants. Custom editing is normally necessary.

9.1.4 The Stop- and Exit-buttons.

The *Stop*-button can be used to abort execution of an AUTO-run. This should be done only in exceptional circumstances. Output-files, if any, will normally be incomplete and should be deleted. Use the *Exit*-button to end a session.

9.2 The Menu Bar.

9.2.1 Equations-button.

This pull-down menu contains the items *Old*, to load an existing equations-file, *New*, to load a model equations-file, and *Demo*, to load a selected demo equations-file. Equations-file names are of the form `xxx.f`. The corresponding constants-file `c.xxx` is also loaded if it exists. The equation name `xxx` remains active until redefined.

9.2.2 Edit-button.

This pull-down menu contains the items *Cut* and *Copy*, to be performed on text in the GUI window highlighted by click-and-drag action of the mouse, and the item *Paste*, which places editor buffer text at the location of the cursor.

9.2.3 Write-button.

This pull-down menu contains the item *Write*, to write the loaded files `xxx.f` and `c.xxx`, by the active equation name, and the item *Write As* to write these files by a selected new name, which then becomes the active name.

9.2.4 Define-button.

Clicking this button will display the full AUTO-constants panel. Most of its text fields can be edited, but some have restricted input values that can be selected with the right mouse button. Some text fields will display a subpanel for entering data. To actually apply changes made in the panel, click the *OK*- or *Apply*-button at the bottom of the panel.

9.2.5 Run-button.

Clicking this button will write the constants-file `c.xxx` and run AUTO. If the equations-file has been edited then it should first be rewritten with the *Write*-button.

9.2.6 Save-button.

This pull-down menu contains the item *Save*, to save the output-files `fort.7`, `fort.8`, `fort.9`, as `b.xxx`, `s.xxx`, `d.xxx`, respectively. Here `xxx` is the active equation name. It also contains the item *Save As*, to save the output-files under another name. Existing data-files with the selected name, if any, will be overwritten.

9.2.7 Append-button.

This pull-down menu contains the item *Append*, to append the output-files `fort.7`, `fort.8`, `fort.9`, to existing data-files `b.xxx`, `s.xxx`, `d.xxx`, respectively. Here `xxx` is the active equation name. It also contains the item *Append To*, to append the output-files to other existing data-files.

9.2.8 Plot-button.

This pull-down menu contains the items *Plot*, to run the plotting program PLAUT for the data-files `b.xxx` and `s.xxx`, where `xxx` is the active equation name, and the item *Name*, to run PLAUT with other data-files.

9.2.9 Files-button.

This pull-down menu contains the item *Restart*, to redefine the restart file. Normally, when restarting from a previously computed solution, the restart data is expected in the file `s.xxx`, where `xxx` is the active equation name. Use the *Restart*-button to read the restart data from another data-file in the immediately following run. The pull-down menu also contains the following items :

- *Copy*, to copy `b.xxx`, `s.xxx`, `d.xxx`, `c.xxx`, to `b.yyy`, `s.yyy`, `d.yyy`, `c.yyy`, resp.;
- *Append*, to append data-files `b.xxx`, `s.xxx`, `d.xxx`, to `b.yyy`, `s.yyy`, `d.yyy`, resp.;
- *Move*, to move `b.xxx`, `s.xxx`, `d.xxx`, `c.xxx`, to `b.yyy`, `s.yyy`, `d.yyy`, `c.yyy`, resp.;
- *Delete*, to delete data-files `b.xxx`, `s.xxx`, `d.xxx`;
- *Clean*, to delete all files of the form `fort.*`, `*.o`, and `*.exe`.

9.2.10 Demos-button.

This pulldown menu contains the items *Select*, to view and run a selected AUTO demo in the demo directory, and *Reset*, to restore the demo directory to its original state. Note that demo files can be copied to the user work directory with the *Equations/Demo*-button.

9.2.11 Misc.-button.

This pulldown menu contains the items *Tek Window* and *VT102 Window*, for opening windows; *Emacs* and *Xedit*, for editing files, and *Print*, for printing the active equations-file **xxx.f**.

9.2.12 Help-button.

This pulldown menu contains the items *AUTO-constants*, for help on AUTO-constants, and *User Manual*, for viewing the user manual; i.e., this document.

9.3 Using the GUI.

AUTO-commands are described in Section 5 and illustrated in the demos. In Table 9.1 we list the main AUTO-commands together with the corresponding GUI button.

<i>@r</i>	<i>Run</i>
<i>@sv</i>	<i>Save</i>
<i>@ap</i>	<i>Append</i>
<i>@p</i>	<i>Plot</i>
<i>@cp</i>	<i>Files/Copy</i>
<i>@mv</i>	<i>Files/Move</i>
<i>@cl</i>	<i>Files/Clean</i>
<i>@dl</i>	<i>Files/Delete</i>
<i>@dm</i>	<i>Equations/Demo</i>

Table 9.1: Command Mode - GUI correspondences.

The AUTO-command *@r xxx yyy* is given in the GUI as follows : click *Files/Restart* and enter *yyy* as data. Then click *Run*. As noted in Section 5, this will run AUTO with the current equations-file **xxx.f** and the current constants-file **c.xxx**, while expecting restart data in **s.yyy**. The AUTO-command *@ap xxx yyy* is given in the GUI by clicking *Files/Append*.

9.4 Customizing the GUI.

9.4.1 Print-button.

The *Misc/Print*-button on the Menu Bar can be customized by editing the file **GuiConsts.h** in directory **auto/07p/include**.

9.4.2 GUI colors.

GUI colors can be customized by creating an X resource file. Two model files can be found in directory **auto/07p/gui**, namely, **Xdefaults.1** and **Xdefaults.2**. To become effective, edit

one of these, if desired, and copy it to `.Xdefaults` in your home directory. Color names can often be found in the system file `/usr/lib/X11/rgb.txt`.

9.4.3 On-line help.

The file `auto/07p/include/GuiGlobal.h` contains on-line help on AUTO-constants and demos. The text can be updated, subject to a modifiable maximum length. On SGI machines this is 10240 bytes, which can be increased, for example, to 20480 bytes, by replacing the line `CC = cc -Wf, -XNl10240 -O` in `auto/07p/gui/Makefile` by `CC = cc -Wf, -XNl20480 -O`. On other machines, the maximum message length is the system defined maximum string literal length.

Chapter 10

Description of AUTO-Constants.

10.1 The AUTO-Constants File.

As described in Section 3, if the equations-file is `xxx.f` then the constants that define the computation are normally expected in the file `c.xxx`. The general format of this file is the same for all AUTO runs. For example, the file `c.cusp` in directory `auto/07p/demos/cusp` is listed below. (The tutorial demo `cusp` is described in detail in Chapter 12.)

```
1 1 0 1          NDIM,IPS,IRS,ILP
2 2 1          NICP,(ICP(I),I=1,NICP)
5 4 3 2 1 0 0 0  NTST,NCOL,IAD,ISP,ISW,IPLT,NBC,NINT
200 -2. 2. 0 100  NMX,RLO,RL1,A0,A1
20 0 2 8 5 3 0    NPR,MXBF,IID,ITMX,ITNW,NWTN,JAC
1.e-6 1.e-6 0.0001 EPSS,EPST,EPSS
0.01 0.005 0.1 1  DS,DSMIN,DSMAX,IADS
0          NTHL,((I,THL(I)),I=1,NTHL)
0          NTHU,((I,THU(I)),I=1,NTHU)
0          NUZR,((I,UZR(I)),I=1,NUZR)
```

The significance of the AUTO-constants, grouped by function, is described in the sections below. Representative demos that illustrate use of the AUTO-constants are also mentioned.

10.2 Problem Constants.

10.2.1 NDIM

Dimension of the system of equations as specified in the user-supplied routine `FUNC`.

10.2.2 NBC

The number of boundary conditions as specified in the user-supplied routine `BCND`.
(Demos `exp`, `kar`.)

10.2.3 NINT

The number of integral conditions as specified in the user-supplied routine ICND.
(Demos `int`, `lin`, `obv`.)

10.2.4 JAC

Used to indicate whether derivatives are supplied by the user or to be obtained by differencing :

- **JAC=0** : No derivatives are given by the user. (Most demos use **JAC=0**.)
- **JAC=1** : Derivatives with respect to state- and problem-parameters are given in the user-supplied routines `FUNC`, `BCND`, `ICND` and `FOPT`, where applicable. This may be necessary for sensitive problems. It is also recommended for computations in which `AUTO` generates an extended system, for example, when `ISW=2`. (For `ISW` see Section 10.8.3.)
(Demos `int`, `dd2`, `obt`, `plp`, `ops`.)

10.3 Discretization Constants.

10.3.1 NTST

The number of mesh intervals used for discretization. **NTST** remains fixed during any particular run, but can be changed when restarting. (For mesh adaption see **IAD** in Section 10.3.3.)
Recommended value of **NTST** : As small as possible to maintain convergence.
(Demos `exp`, `ab`, `spb`.)

10.3.2 NCOL

The number of Gauss collocation points per mesh interval, ($2 \leq \text{NCOL} \leq 7$). **NCOL** remains fixed during any given run, but can be changed when restarting at a previously computed solution. The choice **NCOL=4**, used in most demos, is recommended. If **NDIM** is “large” and the solutions “very smooth” then **NCOL=2** may be appropriate.

10.3.3 IAD

This constant controls the mesh adaption :

- **IAD=0** : Fixed mesh. Normally, this choice should never be used, as it may result in spurious solutions. (Demo `ext`.)
- **IAD>0** : Adapt the mesh every **IAD** steps along the family. Most demos use **IAD=3**, which is the strongly recommended value.

When computing “trivial” solutions to a boundary value problem, for example, when all solution components are constant, then the mesh adaption may fail under certain circumstances, and overflow may occur. In such case, try recomputing the solution family with a fixed mesh (**IAD=0**). Be sure to set **IAD** back to **IAD=3** for computing eventual non-trivial bifurcating solution families.

10.4 Tolerances.

10.4.1 EPSL

Relative convergence criterion for equation parameters in the Newton/Chord method. Most demos use $\text{EPSL}=10^{-6}$ or $\text{EPSL}=10^{-7}$, which is the recommended value range.

10.4.2 EPSU

Relative convergence criterion for solution components in the Newton/Chord method. Most demos use $\text{EPSU}=10^{-6}$ or $\text{EPSU}=10^{-7}$, which is the recommended value range.

10.4.3 EPSS

Relative arclength convergence criterion for the detection of special solutions. Most demos use $\text{EPSS}=10^{-4}$ or $\text{EPSS}=10^{-5}$, which is the recommended value range. Generally, EPSS should be approximately 100 to 1000 times the value of EPSL , EPSU .

10.4.4 ITMX

The maximum number of iterations allowed in the accurate location of special solutions, such as bifurcations, folds, and user output points, by Müller's method with bracketing. The recommended value is $\text{ITMX}=8$, used in most demos.

10.4.5 NWTN

After NWTN Newton iterations the Jacobian is frozen, i.e., AUTO uses full Newton for the first NWTN iterations and the Chord method for iterations $\text{NWTN}+1$ to ITNW . The choice $\text{NWTN}=3$ is strongly recommended and used in most demos. Note that this constant is only effective for ODEs, i.e., for solving the piecewise polynomial collocation equations. For algebraic systems AUTO always uses full Newton.

10.4.6 ITNW

The maximum number of combined Newton-Chord iterations. When this maximum is reached, the step will be retried with half the stepsize. This is repeated until convergence, or until the minimum stepsize is reached. In the latter case the computation of the family is discontinued and a message printed in `fort.9`. The recommended value is $\text{ITNW}=5$, but $\text{ITNW}=7$ may be used for “difficult” problems, for example, demos `spb`, `chu`, `plp`, etc.

10.5 Continuation Step Size.

10.5.1 DS

AUTO uses pseudo-arclength continuation for following solution families. The pseudo-arclength stepsize is the distance between the current solution and the next solution on a family. By default, this distance includes all state variables (or state functions) and all free parameters. The constant **DS** defines the pseudo-arclength stepsize to be used for the first attempted step along any family. (Note that if **IADS**>0 then **DS** will automatically be adapted for subsequent steps and for failed steps.) **DS** may be chosen positive or negative; changing its sign reverses the direction of computation. The relation $\text{DSMIN} \leq |\text{DS}| \leq \text{DSMAX}$ must be satisfied. The precise choice of **DS** is problem-dependent; the demos use a value that was found appropriate after some experimentation.

10.5.2 DSMIN

This is minimum allowable absolute value of the pseudo-arclength stepsize. **DSMIN** must be positive. It is only effective if the pseudo-arclength step is adaptive, i.e., if **IADS**>0. The choice of **DSMIN** is highly problem-dependent; most demos use a value that was found appropriate after some experimentation. See also the discussion in Section 11.2.

10.5.3 DSMAX

The maximum allowable absolute value of the pseudo-arclength stepsize. **DSMAX** must be positive. It is only effective if the pseudo-arclength step is adaptive, i.e., if **IADS**>0. The choice of **DSMAX** is highly problem-dependent; most demos use a value that was found appropriate after some experimentation. See also the discussion in Section 11.2.

10.5.4 IADS

This constant controls the frequency of adaption of the pseudo-arclength stepsize.

- **IADS=0** : Use fixed pseudo-arclength stepsize, i.e., the stepsize will be equal to the specified value of **DS** for every step. The computation of a family will be discontinued as soon as the maximum number of iterations **ITNW** is reached. This choice is not recommended.
(Demo **tim**.)
- **IADS>0** : Adapt the pseudo-arclength stepsize after every **IADS** steps. If the Newton/Chord iteration converges rapidly then $|\text{DS}|$ will be increased, but never beyond **DSMAX**. If a step fails then it will be retried with half the stepsize. This will be done repeatedly until the step is successful or until $|\text{DS}|$ reaches **DSMIN**. In the latter case nonconvergence will be signalled. The strongly recommended value is **IADS=1**, which is used in almost all demos.

10.5.5 NTHL

By default, the pseudo-arclength stepsize includes all state variables (or state functions) and all free parameters. Under certain circumstances one may want to modify the weight accorded to individual parameters in the definition of stepsize. For this purpose, NTHL defines the number of parameters whose weight is to be modified. If NTHL=0 then all weights will have default value 1.0 . If NTHL>0 then one must enter NTHL pairs, *Parameter Index Weight* , with each pair on a separate line.

For example, for the computation of periodic solutions it is recommended that the period not be included in the pseudo-arclength continuation stepsize, in order to avoid period-induced limitations on the stepsize near orbits of infinite period. This exclusion can be accomplished by setting NTHL=1, with, on a separate line, the pair 11 0.0 . Most demos that compute periodic solutions use this option; see for example demo ab.

10.5.6 NTHU

Under certain circumstances one may want to modify the weight accorded to individual state variables (or state functions) in the definition of stepsize. For this purpose, NTHU defines the number of states whose weight is to be modified. If NTHU=0 then all weights will have default value 1.0 . If NTHU>0 then one must enter NTHU pairs, *State Index Weight* , with each pair on a separate line. At present none of the demos use this option.

10.6 Diagram Limits.

There are three ways to limit the computation of a family :

- By appropriate choice of the computational window defined by the constants RL0, RL1, A0, and A1. One should always check that the starting solution lies within this computational window, otherwise the computation will stop immediately at the starting point.
- By specifying the maximum number of steps, NMX.
- By specifying a negative parameter index in the list associated with the constant NUZR; see Section 10.9.4.

10.6.1 NMX

The maximum number of steps to be taken along any family.

10.6.2 RL0

The lower bound on the principal continuation parameter. (This is the parameter which appears first in the ICP list; see Section 10.7.1.).

10.6.3 RL1

The upper bound on the principal continuation parameter.

10.6.4 A0

The lower bound on the principal solution measure. (By default, if IPLT=0, the principal solution measure is the L_2 -norm of the state vector or state vector function. See the AUTO-constant IPLT in Section 10.9.3 for choosing another principal solution measure.)

10.6.5 A1

The upper bound on the principal solution measure.

10.7 Free Parameters.

10.7.1 NICP, ICP

For each equation type and for each continuation calculation there is a typical (“generic”) number of problem parameters that must be allowed to vary, in order for the calculations to be properly posed. The constant NICP indicates how many free parameters have been specified, while the array ICP actually designates these free parameters. The parameter that appears first in the ICP list is called the “principal continuation parameter”. Specific examples and special cases are described below.

10.7.2 Fixed points.

The simplest case is the continuation of a solution family to the system $f(u, p) = 0$, where $f(\cdot, \cdot), u \in \mathbb{R}^n$, cf. Equation (2.1). Such a system arises in the continuation of ODE stationary solutions and in the continuation of fixed points of discrete dynamical systems. There is only one free parameter here, so NICP=1.

As a concrete example, consider Run 1 of demo **ab**, where NICP=1, with ICP(1)=1. Thus, in this run PAR(1) is designated as the free parameter.

10.7.3 Periodic solutions and rotations.

The continuation of periodic solutions and rotations generically requires two parameters, namely, one problem parameter and the period. Thus, in this case NICP=2. For example, in Run 2 of demo **ab** we have NICP=2, with ICP(1)=1 and ICP(2)=11. Thus, in this run, the free parameters are PAR(1) and PAR(11). (Note that AUTO reserves PAR(11) for the period.)

Actually, for periodic solutions, one can set NICP=1 and only specify the index of the free problem parameter, as AUTO will automatically add PAR(11). However, in this case the period will not appear in the screen output and in the **fort.7** output-file.

For fixed period orbits one must set NICP=2 and specify two free problem parameters. For example, in Run 7 of demo **pp2**, we have NICP=2, with PAR(1) and PAR(2) specified as free

problem parameters. The period `PAR(11)` is fixed in this run. If the period is large then such a continuation provides a simple and effective method for computing a locus of homoclinic orbits.

10.7.4 Folds and Hopf bifurcations.

The continuation of folds for algebraic problems and the continuation of Hopf bifurcations requires two free problem parameters, i.e., `NICP=2`. For example, to continue a fold in Run 3 of demo `ab`, we have `NICP=2`, with `PAR(1)` and `PAR(3)` specified as free parameters. Note that one must set `ISW=2` for computing such loci of special solutions. Also note that in the continuation of folds the principal continuation parameter must be the one with respect to which the fold was located.

10.7.5 Folds and period-doublings.

The continuation of folds, for periodic orbits and rotations, and the continuation of period-doubling bifurcations require two free problem parameters plus the free period. Thus, one would normally set `NICP=3`. For example, in Run 6 of demo `pen`, where a locus of period-doubling bifurcations is computed for rotations, we have `NICP=3`, with `PAR(2)`, `PAR(3)`, and `PAR(11)` specified as free parameters. Note that one must set `ISW=2` for computing such loci of special solutions. Also note that in the continuation of folds the principal continuation parameter must be the one with respect to which the fold was located.

Actually, one may set `NICP=2`, and only specify the problem parameters, as AUTO will automatically add the period. For example, in Run 3 of demo `plp`, where a locus of folds is computed for periodic orbits, we have `NICP=2`, with `PAR(4)` and `PAR(1)` specified as free parameters. However, in this case the period will not appear in the screen output and in the `fort.7` output-file.

To continue a locus of folds or period-doublings with fixed period, simply set `NICP=3` and specify three problem parameters, not including `PAR(11)`.

10.7.6 Boundary value problems.

The simplest case is that of boundary value problems where `NDIM=NBC` and where `NINT=0`. Then, generically, one free problem parameter is required for computing a solution family. For example, in demo `exp`, we have `NDIM=NBC=2`, `NINT=0`. Thus `NICP=1`. Indeed, in this demo one free parameter is designated, namely `PAR(1)`.

More generally, for boundary value problems with integral constraints, the generic number of free parameters is `NBC + NINT - NDIM + 1`. For example, in demo `lin`, we have `NDIM=2`, `NBC=2`, and `NINT=1`. Thus `NICP=2`. Indeed, in this demo two free parameters are designated, namely `PAR(1)` and `PAR(3)`.

10.7.7 Boundary value folds.

To continue a locus of folds for a general boundary value problem with integral constraints, set `NICP=NBC+NINT-NDIM+2`, and specify this number of parameter indices to designate the free parameters.

10.7.8 Optimization problems.

In algebraic optimization problems one must set $ICP(1)=10$, as AUTO uses $PAR(10)$ as principal continuation parameter to monitor the value of the objective function. Furthermore, one must designate one free equation parameter in $ICP(2)$. Thus, $NICP=2$ in the first run.

Folds with respect to $PAR(10)$ correspond to extrema of the objective function. In a second run one can restart at such a fold, with an additional free equation parameter specified in $ICP(3)$. Thus, $NICP=3$ in the second run.

The above procedure can be repeated. For example, folds from the second run can be continued in a third run with three equation parameters specified in addition to $PAR(10)$. Thus, $NICP=4$ in the third run.

For a simple example see demo `opt`, where a four-parameter extremum is located. Note that $NICP=5$ in each of the four constants-files of this demo, with the indices of $PAR(10)$ and $PAR(1)$ – $PAR(4)$ specified in ICP . Thus, in the first three runs, there are overspecified parameters. However, AUTO will always use the correct number of parameters. Although the overspecified parameters will be printed, their values will remain fixed.

10.7.9 Internal free parameters.

The actual continuation scheme in AUTO may use additional free parameters that are automatically added. The simplest example is the computation of periodic solutions and rotations, where AUTO automatically adds the period, if not specified. The computation of loci of folds, Hopf bifurcations, and period-doublings also requires additional internal continuation parameters. These will be automatically added, and their indices will be greater than 10.

10.7.10 Parameter overspecification.

The number of specified parameter indices is allowed to be greater than the generic number. In such case there will be “overspecified” parameters, whose values will appear in the screen and `fort.7` output, but which are not part of the continuation process. A simple example is provided by demo `opt`, where the first three runs have overspecified parameters whose values, although constant, are printed.

There is, however, a more useful application of parameter overspecification. In the user-supplied routine `PVLS` one can define solution measures and assign these to otherwise unused parameters. Such parameters can then be overspecified, in order to print them on the screen and in the `fort.7` output. It is important to note that such overspecified parameters must appear at the end of the ICP list, as they cannot be used as true continuation parameters.

For an example of using parameter overspecification for printing user-defined solution measures, see demo `pv1`. This is a boundary value problem (Bratu’s equation) which has only one true continuation parameter, namely $PAR(1)$. Three solution measures are defined in the routine `PVLS`, namely, the L_2 -norm of the first solution component, the minimum of the second component, and the left boundary value of the second component. These solution measures are assigned to $PAR(2)$, $PAR(3)$, and $PAR(4)$, respectively. In the constants-file `c.pv1` we have $NICP=4$, with $PAR(1)$ – $PAR(4)$ specified as parameters. Thus, in this example, $PAR(2)$ – $PAR(4)$

are overspecified. Note that `PAR(1)` must appear first in the `ICP` list; the other parameters cannot be used as true continuation parameters.

10.8 Computation Constants.

10.8.1 ILP

- `ILP=0` : No detection of folds. This choice is recommended.
- `ILP=1` : Detection of folds. To be used if subsequent fold continuation is intended.

10.8.2 ISP

This constant controls the detection of branch points, period-doubling bifurcations, and torus bifurcations.

- `ISP=0` : This setting disables the detection of branch points, period-doubling bifurcations, and torus bifurcations and the computation of Floquet multipliers.
- `ISP=1` : Branch points are detected for algebraic equations, but not for periodic solutions and boundary value problems. Period-doubling bifurcations and torus bifurcations are not located either. However, Floquet multipliers are computed.
- `ISP=2` : This setting enables the detection of all special solutions. For periodic solutions and rotations, the choice `ISP=2` should be used with care, due to potential inaccuracy in the computation of the linearized Poincaré map and possible rapid variation of the Floquet multipliers. The linearized Poincaré map always has a multiplier $z = 1$. If this multiplier becomes inaccurate then the automatic detection of secondary periodic bifurcations will be discontinued and a warning message will be printed in `fort.9`. See also Section 11.4.
- `ISP=3` : Branch points will be detected, but AUTO will not monitor the Floquet multipliers. Period-doubling and torus bifurcations will go undetected. This option is useful for certain problems with non-generic Floquet behavior.

10.8.3 ISW

This constant controls branch switching at branch points for the case of differential equations. Note that branch switching is automatic for algebraic equations.

- `ISW=1` : This is the normal value of `ISW`.
- `ISW=-1` : If `IRS` is the label of a branch point or a period-doubling bifurcation then branch switching will be done. For period doubling bifurcations it is recommended that `NTST` be increased. For examples see Run 2 and Run 3 of demo `lor`, where branch switching is done at period-doubling bifurcations, and Run 2 and Run 3 of demo `bvp`, where branch switching is done at a transcritical branch point.

- **ISW=2** : If **IRS** is the label of a fold, a Hopf bifurcation point, or a period-doubling or torus bifurcation then a locus of such points will be computed. An additional free parameter must be specified for such continuations; see also Section 10.7.

10.8.4 MXBF

This constant, which is effective for algebraic problems only, sets the maximum number of bifurcations to be treated. Additional branch points will be noted, but the corresponding bifurcating families will not be computed. If **MXBF** is positive then the bifurcating families of the first **MXBF** branch points will be traced out in both directions. If **MXBF** is negative then the bifurcating families of the first $|\mathbf{MXBF}|$ branch points will be traced out in only one direction.

10.8.5 IRS

This constant sets the label of the solution where the computation is to be restarted.

- **IRS=0** : This setting is typically used in the first run of a new problem. In this case a starting solution must be defined in the user-supplied routine **STPNT**. For representative examples of analytical starting solutions see demos **ab** and **frc**. For starting from unlabeled numerical data see the *@fc* command (Section 5) and demos **lor** and **pen**.
- **IRS>0** : Restart the computation at the previously computed solution with label **IRS**. This solution is normally expected to be in the current data-file **s.xxx**; see also the *@r* and *@R* commands in Section 5. Various AUTO-constants can be modified when restarting.

10.8.6 IPS

This constant defines the problem type :

- **IPS=0** : An algebraic bifurcation problem. Hopf bifurcations will not be detected and stability properties will not be indicated in the **fort.7** output-file.
- **IPS=1** : Stationary solutions of ODEs with detection of Hopf bifurcations. The sign of PT, the point number, in **fort.7** is used to indicate stability : $-$ is stable , $+$ is unstable. (Demo **ab**.)
- **IPS=-1** : Fixed points of the discrete dynamical system $u^{(k+1)} = f(u^{(k)}, p)$, with detection of Hopf bifurcations. The sign of PT in **fort.7** indicates stability : $-$ is stable , $+$ is unstable. (Demo **dd2**.)
- **IPS=-2** : Time integration using implicit Euler. The AUTO-constants **DS**, **DSMIN**, **DSMAX**, and **ITNW**, **NWTN** control the stepsize. In fact, pseudo-arclength is used for “continuation in time”. Note that the time discretization is only first order accurate, so that results should be carefully interpreted. Indeed, this option has been included primarily for the detection of stationary solutions, which can then be entered in the user-supplied routine **STPNT**. (Demo **ivp**.)

- **IPS=2** : Computation of periodic solutions. Starting data can be a Hopf bifurcation point (Run 2 of demo **ab**), a periodic orbit from a previous run (Run 4 of demo **pp2**), an analytically known periodic orbit (Run 1 of demo **frc**), or a numerically known periodic orbit (Demo **lor**). The sign of PT in **fort.7** is used to indicate stability : $-$ is stable , $+$ is unstable or unknown.
- **IPS=4** : A boundary value problem. Boundary conditions must be specified in the user-supplied routine **BCND** and integral constraints in **ICND**. The AUTO-constants **NBC** and **NINT** must be given correct values. (Demos **exp**, **int**, **kar**.)
- **IPS=5** : Algebraic optimization problems. The objective function must be specified in the user-supplied routine **FOPT**. (Demo **opt**.)
- **IPS=7** : A boundary value problem with computation of Floquet multipliers. This is a very special option; for most boundary value problems one should use **IPS=4**. Boundary conditions must be specified in the user-supplied routine **BCND** and integral constraints in **ICND**. The AUTO-constants **NBC** and **NINT** must be given correct values.
- **IPS=9** : This option is used in connection with the HomCont algorithms described in Chapters 20-26 for the detection and continuation of homoclinic bifurcations. (Demos **san**, **mtn**, **kpr**, **cir**, **she**, **rev**.)
- **IPS=11** : Spatially uniform solutions of a system of parabolic PDEs, with detection of traveling wave bifurcations. The user need only define the nonlinearity (in routine **FUNC**), initialize the wave speed in **PAR(10)**, initialize the diffusion constants in **PAR(15,16,...)**, and set a free equation parameter in **ICP(1)**. (Run 2 of demo **wav**.)
- **IPS=12** : Continuation of traveling wave solutions to a system of parabolic PDEs. Starting data can be a Hopf bifurcation point from a previous run with **IPS=11**, or a traveling wave from a previous run with **IPS=12**. (Run 3 and Run 4 of demo **wav**.)
- **IPS=14** : Time evolution for a system of parabolic PDEs subject to periodic boundary conditions. Starting data may be solutions from a previous run with **IPS=12** or **14**. Starting data can also be specified in **STPNT**, in which case the wave length must be specified in **PAR(11)**, and the diffusion constants in **PAR(15,16,...)**. AUTO uses **PAR(14)** for the time variable. **DS**, **DSMIN**, and **DSMAX** govern the pseudo-arclength continuation in the space-time variables. Note that the time discretization is only first order accurate, so that results should be carefully interpreted. Indeed, this option is mainly intended for the detection of stationary waves. (Run 5 of demo **wav**.)
- **IPS=15** : Optimization of periodic solutions. The integrand of the objective functional must be specified in the user supplied routine **FOPT**. Only **PAR(1-9)** should be used for problem parameters. **PAR(10)** is the value of the objective functional, **PAR(11)** the period, **PAR(12)** the norm of the adjoint variables, **PAR(14)** and **PAR(15)** are internal optimality variables. **PAR(21-29)** and **PAR(31)** are used to monitor the optimality functionals associated with the problem parameters and the period. Computations can be started at a solution computed with **IPS=2** or **IPS=15**. For a detailed example see demo **ops**.

- **IPS=16** : This option is similar to **IPS=14**, except that the user supplies the boundary conditions. Thus this option can be used for time-integration of parabolic systems subject to user-defined boundary conditions. For examples see the first runs of demos **pd1**, **pd2**, and **bru**. Note that the space-derivatives of the initial conditions must also be supplied in the user supplied routine **STPNT**. The initial conditions must satisfy the boundary conditions. This option is mainly intended for the detecting stationary solutions.
- **IPS=17** : This option can be used to continue stationary solutions of parabolic systems obtained from an evolution run with **IPS=16**. For examples see the second runs of demos **pd1** and **pd2**.

10.9 Output Control.

10.9.1 NPR

This constant can be used to regularly write **fort.8** plotting and restart data. IF **NPR**>0 then such output is written every **NPR** steps. IF **NPR**=0 or if **NPR**≥**NMX** then no such output is written. Note that special solutions, such as branch points, folds, end points, etc., are always written in **fort.8**. Furthermore, one can specify parameter values where plotting and restart data is to be written; see Section 10.9.4. For these reasons, and to limit the output volume, it is recommended that **NPR** output be kept to a minimum.

10.9.2 IID

This constant controls the amount of diagnostic output printed in **fort.9** : the greater **IID** the more detailed the diagnostic output.

- **IID=0** : Minimal diagnostic output. This setting is not recommended.
- **IID=2** : Regular diagnostic output. This is the recommended value of **IID**.
- **IID=3** : This setting gives additional diagnostic output for algebraic equations, namely the Jacobian and the residual vector at the starting point. This information, which is printed at the beginning of **fort.9**, is useful for verifying whether the starting solution in **STPNT** is indeed a solution.
- **IID=4** : This setting gives additional diagnostic output for differential equations, namely the reduced system and the associated residual vector. This information is printed for every step and for every Newton iteration, and should normally be suppressed. In particular it can be used to verify whether the starting solution is indeed a solution. For this purpose, the stepsize **DS** should be small, and one should look at the residuals printed in the **fort.9** output-file. (Note that the first residual vector printed in **fort.9** may be identically zero, as it may correspond to the computation of the starting direction. Look at the second residual vector in such case.) This residual vector has dimension **NDIM**+**NBC**+**NINT**+1, which accounts for the **NDIM** differential equations, the **NBC** boundary conditions, the **NINT**

user-defined integral constraints, and the pseudo-arclength equation. For proper interpretations of these data one may want to refer to the solution algorithm for solving the collocation system, as described in Doedel, Keller & Kernévez (1991b).

- **IID=5** : This setting gives very extensive diagnostic output for differential equations, namely, debug output from the linear equation solver. This setting should not normally be used as it may result in a huge **fort.9** file.

10.9.3 IPLT

This constant allows redefinition of the principal solution measure, which is printed as the second (real) column in the screen output and in the **fort.7** output-file :

- If **IPLT** = 0 then the L_2 -norm is printed. Most demos use this setting. For algebraic problems, the standard definition of L_2 -norm is used. For differential equations, the L_2 -norm is defined as

$$\sqrt{\int_0^1 \sum_{k=1}^{NDIM} U_k(x)^2 dx} .$$

Note that the interval of integration is $[0, 1]$, the standard interval used by AUTO. For periodic solutions the independent variable is transformed to range from 0 to 1, before the norm is computed. The AUTO-constants **THL(*)** and **THU(*)** (see Section 10.5.5 and Section 10.5.6) affect the definition of the L_2 -norm.

- If $0 < \text{IPLT} \leq \text{NDIM}$ then the maximum of the **IPLT**'th solution component is printed.
- If $-\text{NDIM} \leq \text{IPLT} < 0$ then the minimum of the **IPLT**'th solution component is printed. (Demo **fsh.**)
- If $\text{NDIM} < \text{IPLT} \leq 2*\text{NDIM}$ then the integral of the $(\text{IPLT}-\text{NDIM})$ 'th solution component is printed. (Demos **exp**, **lor.**)
- If $2*\text{NDIM} < \text{IPLT} \leq 3*\text{NDIM}$ then the L_2 -norm of the $(\text{IPLT}-\text{NDIM})$ 'th solution component is printed. (Demo **frc.**)

Note that for algebraic problems the maximum and the minimum are identical. Also, for ODEs the maximum and the minimum of a solution component are generally much less accurate than the L_2 -norm and component integrals. Note also that the routine **PVLS** provides a second, more general way of defining solution measures; see Section 10.7.10.

10.9.4 NUZR

This constant allows the setting of parameter values at which labeled plotting and restart information is to be written in the **fort.8** output-file. Optionally, it also allows the computation to terminate at such a point.

- Set **NUZR=0** if no such output is needed. Many demos use this setting.

- If `NUZR>0` then one must enter `NUZR` pairs, *Parameter-Index* *Parameter-Value* , with each pair on a separate line, to designate the parameters and the parameter values at which output is to be written. For examples see demos `exp`, `int`, and `fsh`.
- If such a parameter index is preceded by a minus sign then the computation will terminate at such a solution point. (Demos `pen` and `bru`.)

Note that `fort.8` output can also be written at selected values of overspecified parameters. For an example see demo `pv1`. For details on overspecified parameters see Section 10.7.10.

10.10 Quick reference

NDIM	Problem dimension
IPS	Problem type; 0=AE, 1=FP(ODEs), -1=FP(maps), 2=PO, -2=IVP, 4=BVP, 7=BVP with Floquet multipliers, 5=algebraic optimization problem, 15=optimization of periodic solutions
IRS	Start solution label
ILP	Fold detection; 1=on, 0=off
NICP	Continuation parameters
NTST	# mesh intervals
NCOL	# collocation points
IAD	Mesh adaption every IAD steps; 0=off
ISP	Bifurcation detection; 0=off, 1=BP(FP), 3=BP(PO,BVP), 2=all
ISW	Branch switching; 1=normal, -1=switch branch (BP, HB, PD), 2=switch to two-parameter continuation (LP, HB, TR)
IPLT	Select principal solution measure
NBC	# boundary conditions
NINT	# integral conditions
NMX	Maximum number of steps
RLO, RL1	Parameter interval $RL0 \leq \lambda \leq RL1$
A0, A1	Interval of principal solution measure $A0 \leq \ \cdot\ \leq A1$
NPR	Print and save restart data every NPR steps
MXBF	Automatic branch switching for the first MXBF bifurcation points if IPS=0, 1
IID	Control diagnostic output; 1=little, 2=normal, 4=extensive
ITMX	Maximum # of iterations for locating special solutions/points
ITNW	Maximum # of correction steps
NWTN	Corrector uses full newton for NWTN steps
JAC	User defines derivatives; 0=no, 1=yes
EPSL	Convergence criterion for equation parameters
EPSU	Convergence criterion for solution components
EPSS	Convergence criterion for special solutions/points
DS	Start step size
DSMIN, DSMAX	Step size interval $DSMIN \leq h \leq DSMAX$
IADS	Step size adaption every IADS steps; 0=off
NTHL, NTHU	list of parameter and solution weights
NUZR	list of values for user defined output

Chapter 11

Notes on Using AUTO.

11.1 Restrictions on the Use of PAR.

The array `PAR` in the user-supplied routines is available for equation parameters that the user wants to vary at some point in the computations. In any particular computation the free parameter(s) must be designated in `ICP`; see Section 10.7. The following restrictions apply :

- The maximum number of parameters, `NPARX` in `auto/07p/include/auto.h`, has pre-defined value `NPARX=36`. `NPARX` should not normally be increased and it should never be decreased. Any increase of `NPARX` must be followed by recompilation of `AUTO`.
- Generally one should only use `PAR(1)–PAR(9)` for equation parameters, as `AUTO` may need the other components internally.

11.2 Efficiency.

In `AUTO`, efficiency has at times been sacrificed for generality of programming. This applies in particular to computations in which `AUTO` generates an extended system, for example, computations with `ISW=2`. However, the user has significant control over computational efficiency, in particular through judicious choice of the `AUTO`-constants `DS`, `DSMIN`, and `DSMAX`, and, for ODEs, `NTST` and `NCOL`. Initial experimentation normally suggests appropriate values.

Slowly varying solutions to ODEs can often be computed with remarkably small values of `NTST` and `NCOL`, for example, `NTST=5`, `NCOL=2`. Generally, however, it is recommended to set `NCOL=4`, and then to use the “smallest” value of `NTST` that maintains convergence.

The choice of the pseudo-arclength stepsize parameters `DS`, `DSMIN`, and `DSMAX` is highly problem dependent. Generally, `DSMIN` should not be taken too small, in order to prevent excessive step refinement in case of non-convergence. It should also not be too large, in order to avoid instant non-convergence. `DSMAX` should be sufficiently large, in order to reduce computation time and amount of output data. On the other hand, it should be sufficiently small, in order to prevent stepping over bifurcations without detecting them. For a given equation, appropriate values of these constants can normally be found after some initial experimentation.

The constants `ITNW`, `NWTN`, `THL`, `EPSU`, `EPSL`, `EPSS` also affect efficiency. Understanding their significance is therefore useful; see Section 10.4 and Section 10.5. Finally, it is recommended

that initial computations be done with `ILP=0`; no fold detection; and `ISP=1`; no bifurcation detection for ODEs.

11.3 Correctness of Results.

AUTO-computed solutions to ODEs are almost always structurally correct, because the mesh adaption strategy, if `IAD>0`, safeguards to some extent against spurious solutions. If these do occur, possibly near infinite-period orbits, the unusual appearance of the solution family typically serves as a warning. Repeating the computation with increased `NTST` is then recommended.

11.4 Bifurcation Points and Folds.

It is recommended that the detection of folds and bifurcation points be initially disabled. For example, if an equation has a “vertical” solution family then AUTO may try to locate one fold after another.

Generally, degenerate bifurcations cannot be detected. Furthermore, bifurcations that are close to each other may not be noticed when the pseudo-arclength step size is not sufficiently small. Hopf bifurcation points may go unnoticed if no clear crossing of the imaginary axis takes place. This may happen when there are other real or complex eigenvalues near the imaginary axis and when the pseudo-arclength step is large compared to the rate of change of the critical eigenvalue pair. A typical case is a Hopf bifurcation close to a fold. Similarly, Hopf bifurcations may go undetected if switching from real to complex conjugate, followed by crossing of the imaginary axis, occurs rapidly with respect to the pseudo-arclength step size. Secondary periodic bifurcations may not be detected for similar reasons. In case of doubt, carefully inspect the contents of the diagnostics file `fort.9`.

11.5 Floquet Multipliers.

AUTO extracts an approximation to the linearized Poincaré map from the Jacobian of the linearized collocation system that arises in Newton’s method. This procedure is very efficient; the map is computed at negligible extra cost. The linear equations solver of AUTO is described in Doedel, Keller & Kernévez (1991*b*). The actual Floquet multiplier solver was written by Fairgrieve (1994). For a detailed description of the algorithm see Fairgrieve & Jepson (1991).

For periodic solutions, the exact linearized Poincaré map always has a multiplier $z = 1$. A good accuracy check is to inspect this multiplier in the diagnostics output-file `fort.9`. If this multiplier becomes inaccurate then the automatic detection of potential secondary periodic bifurcations (if `ISP=2`) is discontinued and a warning is printed in `fort.9`. It is strongly recommended that the contents of this file be habitually inspected, in particular to verify whether solutions labeled as BP or TR (cf. Table 6.1) have indeed been correctly classified.

11.6 Memory Requirements.

The run-time memory requirements depend on the values the user set in the constants file and are roughly proportional to the value $NTST \times (NDIM \times (NCOL + 1) + NBC) \times (NDIM \times NCOL + NINT + 1)$.

Chapter 12

AUTO Demos : Tutorial.

12.1 Introduction.

The directory `auto/07p/demos` has a large number of subdirectories, for example `ab`, `pp2`, `exp`, etc., each containing all necessary files for certain illustrative calculations. Each subdirectory, say `xxx`, corresponds to a particular equation and contains one equations-file `xxx.{f90,f,c}` and one or more constants-files `c.xxx.i`, one for each successive run of the demo. To see how the equations have been programmed, inspect the equations-file. To understand in detail how AUTO is instructed to carry out a particular task, inspect the appropriate constants-file. In this chapter we describe the tutorial demo `cusps` in detail. A brief description of other demos is given in later chapters.

12.2 `cusps` : A Tutorial Demo.

This demo illustrates the computation of stationary solutions, locating saddle-node bifurcations of these solutions, and the continuation of a saddle-node bifurcation in two parameters. The `cusps` normal form equation is given by

$$\dot{x} = \mu + \lambda x - x^3. \quad (12.1)$$

12.3 Copying the Demo Files.

The commands listed in Table 12.1 will copy the demo files to your work directory.

Unix-COMMAND	ACTION
<code>auto</code>	start the AUTO-07p Command Line User Interface
AUTO -COMMAND	ACTION
<code>cd</code>	go to main directory (or other directory).
<code>! mkdir cusps</code>	create an empty work directory. Note: the <code>'!'</code> is used to signify a command which is sent to the shell.
<code>cd cusps</code>	change to the work directory.
<code>demo('cusps')</code>	copy the demo files to the work directory.

Table 12.1: Copying the demo `cusps` files.

Typing `ls` reveals the existence of 5 files:

1. `cusps.f90` : This file contains the differential equations and the initial values. If you inspect it, you will see that only two routines are used. The subroutine `FUNC` specifies the actual differential equation. The routine `STPNT` gives AUTO the initial values of `PAR(1) = λ` and `PAR(2) = μ` , which are 1.0 and 0.0, and the initial value of `x`, which is 0. For your own models you would generally copy another equation file and then only change the pieces that actually define the equation.

2. `c.cusp` : The initial computational constants are stored in this file. Most importantly, you see that the dimension of the problem (NDIM) is set to 1, and the problem type IPS is set to 1 to specify continuation of a stationary solution. The constants given by ICP specify the parameters that are used for the continuation. In this case these are 2 for μ and 1 for λ . Since initially we only really continue in one parameter (μ), the second parameter λ is *overspecified*. Another important constant is the initial step size DS: as it is positive, we initially continue in the positive μ direction.
3. `c.cusp.1` : A copy of `c.cusp`, for convenience.
4. `cusp.auto` : A script with Python CLUI commands that steer the calculation.
5. `plaut04.rc` : A file that contains default settings for the 3-dimensional plotting tool PLAUT04.

12.4 Executing all Runs Automatically.

To execute all prepared runs of demo `cusp`, simply type the command given in Table 12.2.

AUTO -COMMAND	ACTION
<code>demofile('cusp.auto')</code>	execute all runs of demo <code>ab</code> interactively

Table 12.2: Executing all runs of demo `cusp`.

The command in Table 12.2 begins a tutorial which will proceed one step each time the user presses a key. Each step consists of a single AUTO command preceded by instructions as to what action the command performs. The tutorial script `cusp.auto` performs the demo by reading in a single AUTO constants file and then interactively modifying it to perform each of the demo. The essential commands in `cusp.auto` are given in Table 12.4.

Note that there are four separate runs, where each `run` command (`r` is an abbreviation) performs a run. In the first run, a branch of stationary solutions is traced out. Along it, one fold (LP) (limit point, or in this case, a saddle-node bifurcation) is located. The free parameter is μ . The other parameter λ remains fixed in this run. Note also that only special, labeled solution points are printed on the screen. More detailed results are saved in the data-files `b.mu`, `s.mu`, and `d.mu`.

The second run does the same thing but now in the negative direction of μ , i.e., backwards instead of forwards. The backwards continuation is appended to the forwards continuation in the data-files. Afterwards we perform a relabelling to make sure that we have unique labels for each special solution.

The results are then plotted on the screen. Pressing the enter key at the command line causes an automatic μ vs. x display that shows the two fold points at labels 2 and 7.

In the third run, the fold detected in the first run is followed in the two parameters μ and λ . The commands that accomplish this must change a few constants of the constant file: the first constant is ISW, that must be set to 2 to cause a two-parameter continuation.

The second constant is the starting label for the file `s.mu` that we are starting from. We know that 2 is a good label. However, we did not know this number in advance, and moreover, in sensitive cases, it can be different on different computer types. Another way to specify the starting label is to use the `splabs` Python function: in this case `splabs('mu','LP')` returns the list of 'LP' labels for `b.mu`, and `s.mu`, and so the first one, `splabs('mu','LP')[0]` denotes label 2.

The fourth run continues this branch in opposite direction. The detailed results of these continuations are accumulated in the data-files `b.cusp`, `s.cusp`, and `d.cusp`. Finally, a plot of the cusp is produced.

The numerical results are given below in somewhat abbreviated form. Some differences in output are to be expected on different machines. This does not mean that the results have different accuracy, but simply that arithmetic differences have accumulated from step to step, possibly leading to different step size decisions.

Next, reset the work directory, by typing the command given in Table 12.3.

AUTO -COMMAND	ACTION
<code>cl()</code>	remove temporary files of demo <code>cusp</code>
<code>dl('mu')</code>	remove 'mu' data-files of demo <code>cusp</code>
<code>dl('cusp')</code>	remove 'cusp' data-files of demo <code>cusp</code>

Table 12.3: Cleaning the demo `cusp` work directory.

Run forwards

BR	PT	TY	LAB	PAR(2)	L2-NORM	U(1)	PAR(1)
1	1	EP	1	0.00000E+00	0.00000E+00	0.00000E+00	1.00000E+00
1	14	LP	2	3.84900E-01	5.77360E-01	-5.77360E-01	1.00000E+00
1	20		3	1.26582E-01	9.29410E-01	-9.29410E-01	1.00000E+00
1	40		4	-1.38347E+00	1.40803E+00	-1.40803E+00	1.00000E+00
1	47	EP	5	-2.07212E+00	1.53340E+00	-1.53340E+00	1.00000E+00

Run backwards

BR	PT	TY	LAB	PAR(2)	L2-NORM	U(1)	PAR(1)
1	1	EP	1	0.00000E+00	0.00000E+00	0.00000E+00	1.00000E+00
1	14	LP	2	-3.84900E-01	5.77360E-01	5.77360E-01	1.00000E+00
1	20		3	-1.26582E-01	9.29410E-01	9.29410E-01	1.00000E+00
1	40		4	1.38347E+00	1.40803E+00	1.40803E+00	1.00000E+00
1	47	EP	5	2.07212E+00	1.53340E+00	1.53340E+00	1.00000E+00

Forward continuation of the first fold in two parameters

BR	PT	TY	LAB	PAR(2)	L2-NORM	U(1)	PAR(1)
2	20		11	5.42543E-02	3.00470E-01	-3.00470E-01	2.70847E-01
2	40		12	-9.99228E-02	3.68308E-01	3.68308E-01	4.06953E-01


```

2    60      13 -5.89228E-01  6.65403E-01  6.65403E-01  1.32828E+00
2    80      14 -1.71622E+00  9.50271E-01  9.50271E-01  2.70904E+00
2    85  EP   15 -2.06374E+00  1.01051E+00  1.01051E+00  3.06341E+00

```

Backward continuation of the fold in two parameters

```

BR    PT  TY  LAB    PAR(2)      L2-NORM      U(1)      PAR(1)
2     20      11  1.09209E+00  8.17354E-01 -8.17354E-01  2.00420E+00
2     34  EP   12  2.02776E+00  1.00461E+00 -1.00461E+00  3.02770E+00

```

The CLUI was used to generate the constants file at runtime. In the example below, the constant file `c.cusp` will be read in, and the CLUI will be used to make the appropriate changes to perform the calculation.

AUTO -COMMAND	ACTION
<code>ld('cusp')</code>	load the problem definition <code>cusp</code>
<code>r(sv='mu')</code>	execute the run and save the results in the files <code>b.mu</code> , <code>s.mu</code> , and <code>d.mu</code>
<code>r(DS='-',ap='mu')</code>	execute the run backwards and append the results to the above files
<code>rl('mu')</code>	relabel solutions in <code>b.mu</code> and <code>s.mu</code>
<code>ld(s='mu',ISW=2,IRS=splabs('mu','LP')[0])</code>	use <code>s.mu</code> as the solution file to start from, change <code>ISW</code> to 2, and change <code>IRS</code> to select the first fold in <code>b.mu</code> .
<code>r(sv='cusp')</code>	execute the third run of demo <code>cusp</code>
<code>r(DS='-',ap='cusp')</code>	execute the fourth run of demo <code>cusp</code>

Table 12.4: Selected runs of demo `ab`.

12.5 Plotting the Results with AUTO .

The bifurcation diagram computed in the runs above was stored in the files `b.mu` and `b.cusp`, while each labeled solution is fully stored in `s.mu` and `s.cusp`. To use AUTO to graphically inspect these data-files. type the AUTO -command given in Table 12.5. The saved plots are shown in Figure 12.1 and in Figure 12.2.

Figure 12.1 shows the bifurcation diagrams for the first run, and Figure 12.2 for the second run.

The plotting window consists of a menubar at the top, a plotting area, and a control panel with four control widgets at the bottom. By default the first two columns in the bifurcation diagram output are plotted against each other. To obtain a μ versus x bifurcation diagram you need to plot column 0 versus column 1. You can do that by changing the “Y” box to say “[2]”, either by typing it there, by using the menu obtained by clicking the downwards facing triangle or by using a scripted command as used in `cusp.auto`. You can also change the mode of the

plotting tool from “bifurcation” to “solution”. This is accomplished by clicking on the widget marked “Type” on the bottom control panel and setting it from “bifurcation” to “solution”. In the plotting window will appear a plot of the first labeled solution, in this case just a point. You can plot all points by changing the “Label” to “[1,2,3,4,5,6,7,8,9,10]”.

The plotting tool can also be used to create Postscript files from plots by selecting the “File” on the menubar and then selecting the “Save Postscript...” from the drop down menu. This will bring up a dialog into which the user can enter the filename of the postscript file to save the plot in. Further information on the plotting tool can be found in Section 4.11.

AUTO -COMMAND	ACTION
<code>plot("mu")</code>	run AUTO to graph the contents of <code>b.mu</code> and <code>s.mu</code> ;

Table 12.5: Command for plotting the files `b.ab` and `s.ab`.

12.6 Plotting the Results with AUTO in 3D.

Whilst not very useful for this simple example, you can also plot your results in 3D, using the `plot3` command (PLAUT04), for example `plot3('mu')`. Unlike the 2D tool, this shows the stable and unstable parts: blue is stable, and red is unstable. You can also spin the bifurcation diagram around and zoom in using the mouse.

12.7 Exporting the Results for different plotters.

It is often useful to use other plotting programs or general-purpose tools to work with AUTO’s data. The “writeRawFilename” method (see also the CLUI chapter) can be used for this. In this tutorial we can for instance export the bifurcation diagram using `dg('cusp').writeRawFilename('cusp.dat')`, and then use the command `plot 'cusp.dat' using 1:4 wi li` to plot the bifurcation diagram in GNUPlot.

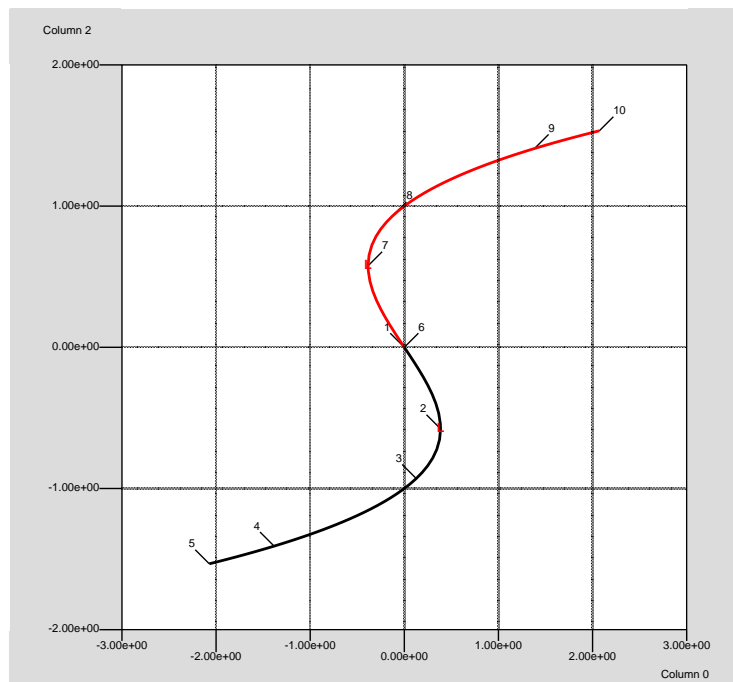


Figure 12.1: The first bifurcation diagram of demo cusp.

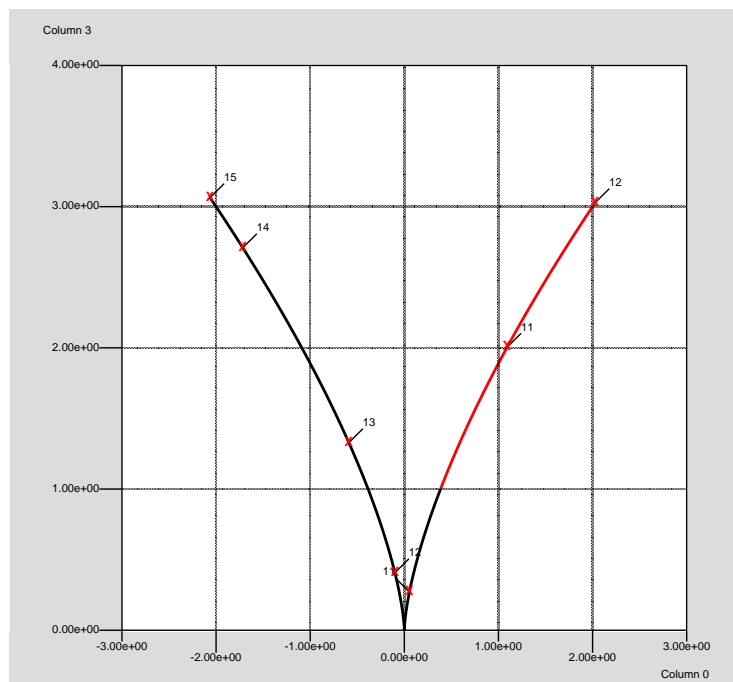


Figure 12.2: The second bifurcation diagram of demo cusp.

12.8 ab : A Programmed Demo.

This demo illustrates the computation of stationary solutions, Hopf bifurcations and periodic solutions. The equations, that model an $A \rightarrow B$ reaction, are those from Uppal, Ray & Poore (1974), namely

$$\begin{aligned}u_1' &= -u_1 + p_1(1 - u_1)e^{u_2}, \\u_2' &= -u_2 + p_1p_2(1 - u_1)e^{u_2} - p_3u_2.\end{aligned}\tag{12.2}$$

This demo is full scripted, see Table 12.6.

AUTO -COMMAND	ACTION
<code>! mkdir ab</code>	create an empty work directory
<code>cd ab</code>	change directory
<code>demo('ab')</code>	copy the demo files to the work directory
<code>execfile('ab.auto')</code>	run the demo

Table 12.6: Commands for running demo **ab**.

If you look at the file **ab.auto** you see that the script computes a stationary solution family for certain values of p_2 , and that a periodic orbit family is computed for each Hopf bifurcation that was found in the stationary solution families.

Chapter 13

AUTO Demos : Fixed points.

13.1 enz : Stationary Solutions of an Enzyme Model.

The equations, that model a two-compartment enzyme system (Kernévez (1980)), are given by

$$\begin{aligned} s'_1 &= (s_0 - s_1) + (s_2 - s_1) - \rho R(s_1), \\ s'_2 &= (s_0 + \mu - s_2) + (s_1 - s_2) - \rho R(s_2), \end{aligned} \tag{13.1}$$

where

$$R(s) = \frac{s}{1 + s + \kappa s^2}.$$

The free parameter is s_0 . Other parameters are fixed. This equation is also considered in Doedel, Keller & Kernévez (1991a).

AUTO -COMMAND	ACTION
<code>! mkdir enz</code>	create an empty work directory
<code>cd enz</code>	change directory
<code>demo('enz')</code>	copy the demo files to the work directory
<code>ld('enz')</code>	load the problem definition
<code>run(c='enz.1')</code>	compute stationary solution families
<code>sv('enz')</code>	save output-files as <code>b.enz</code> , <code>s.enz</code> , <code>d.enz</code>

Table 13.1: Commands for running demo `enz`.

13.2 dd2 : Fixed Points of a Discrete Dynamical System.

This demo illustrates the computation of a solution family and its bifurcating families for a discrete dynamical system. Also illustrated is the continuation of Naimark-Sacker (or Hopf) bifurcations. The equations, a discrete predator-prey system, are

$$\begin{aligned} u_1^{k+1} &= p_1 u_1^k (1 - u_1^k) - p_2 u_1^k u_2^k, \\ u_2^{k+1} &= (1 - p_3) u_2^k + p_2 u_1^k u_2^k. \end{aligned} \quad (13.2)$$

In the first run p_1 is free. In the second run, both p_1 and p_2 are free. The remaining equation parameter, p_3 , is fixed in both runs.

AUTO -COMMAND	ACTION
<code>! mkdir dd2</code>	create an empty work directory
<code>cd dd2</code>	change directory
<code>demo('dd2')</code>	copy the demo files to the work directory
<code>ld('dd2')</code>	load the problem definition
<code>run(c='dd2.1')</code>	1st run; fixed point solution branches
<code>sv('dd2')</code>	save output-files as <code>b.dd2</code> , <code>s.dd2</code> , <code>d.dd2</code>
<code>run(c='dd2.2',s='dd2')</code>	2nd run; a locus of Naimark-Sacker bifurcations. Constants changed : <code>IRS</code> , <code>ISW</code>
<code>sv('ns')</code>	save output-files as <code>b.ns</code> , <code>s.ns</code> , <code>d.ns</code>

Table 13.2: Commands for running demo `dd2`.

Chapter 14

AUTO Demos : Periodic solutions.

14.1 lrz : The Lorenz Equations.

This demo computes two symmetric homoclinic orbits in the Lorenz equations

$$\begin{aligned} u_1' &= p_3(u_2 - u_1), \\ u_2' &= p_1 u_1 - u_2 - u_1 u_3, \\ u_3' &= u_1 u_2 - p_2 u_3. \end{aligned} \tag{14.1}$$

Here p_1 is the free parameter, and $p_2 = 8/3$, $p_3 = 10$. The two homoclinic orbits correspond to the final, large period orbits on the two periodic solution families.

AUTO -COMMAND	ACTION
<code>! mkdir lrz</code> <code>cd lrz</code> <code>demo('lrz')</code>	create an empty work directory change directory copy the demo files to the work directory
<code>ld('lrz')</code> <code>run(c='lrz.1')</code> <code>sv('lrz')</code>	load the problem definition compute stationary solutions save output-files as <code>b.lrz</code> , <code>s.lrz</code> , <code>d.lrz</code>
<code>run(c='lrz.2',s='lrz')</code> <code>ap('lrz')</code>	compute periodic solutions; the final orbit is near-homoclinic. Constants changed : <code>IPS</code> , <code>IRS</code> , <code>NICP</code> , <code>ICP</code> , <code>NMX</code> , <code>NPR</code> , <code>DS</code> append the output-files to <code>b.lrz</code> , <code>s.lrz</code> , <code>d.lrz</code>
<code>run(c='lrz.3',s='lrz')</code> <code>ap('lrz')</code>	compute the symmetric periodic solution family. Constants changed : <code>IRS</code> append the output-files to <code>b.lrz</code> , <code>s.lrz</code> , <code>d.lrz</code>

Table 14.1: Commands for running demo `lrz`.

14.2 abc : The $A \rightarrow B \rightarrow C$ Reaction.

This demo illustrates the computation of stationary solutions, Hopf bifurcations and periodic solutions in the $A \rightarrow B \rightarrow C$ reaction (Doedel & Heinemann (1983)).

$$\begin{aligned} u_1' &= -u_1 + p_1(1 - u_1)e^{u_3}, \\ u_2' &= -u_2 + p_1e^{u_3}(1 - u_1 - p_5u_2), \\ u_3' &= -u_3 - p_3u_3 + p_1p_4e^{u_3}(1 - u_1 + p_2p_5u_2), \end{aligned} \tag{14.2}$$

with $p_2 = 1$, $p_3 = 1.55$, $p_4 = 8$, and $p_5 = 0.04$. The free parameter is p_1 .

The equations, as programmed in the equations-file `abc.f`, appear in Table 14.2. The starting point, an equilibrium of the equations, is also defined in the equations-file `abc.f`, as shown in Table 14.3. (The equations-file `abc.f` also contains the skeletons of some other routines, which must be supplied, but which are not used in this application.)

In the constants-file (`c.abc.1`) for the first run, as shown in Table 14.4, we note the following:

- `IPS=1` : a family of stationary solutions is computed.
- `IRS=0` : the starting point defined in `STPNT` is to be used (see Table 14.3).
- `ICP(1)=1` : the continuation parameter is `PAR(1)`
- `NUZR=1` : there is one user output point, namely at `PAR(1)=0.4`. Moreover, since the index ("–1") in the last line of the constants-file `c.abc.1` is negative, the calculation will terminate when the calculation reaches the value `PAR(1)=0.4`.

In the constants-file (`c.abc.2`) for the second run, as shown in Table 14.5, we note that:

- `IPS=2` : a family of periodic solutions is computed.
- `IRS=2` : the starting point is the solution with label 2, (a Hopf bifurcation point), to be read from the solutions-file (here `s.abc`).
- `NICP=2` : there are two continuation parameters (namely `PAR(1)`, and the period, `PAR(11)`).
- `NUZR=1` : there is one user output point, now at `PAR(1)=0.25`, where the calculation is to terminate, since the index ("–1") is negative.

	SUBROUTINE FUNC(NDIM,U,ICP,PAR,IJAC,F,DFDU,DFDP)
C	-----
C	
	IMPLICIT DOUBLE PRECISION (A-H,O-Z)
	DIMENSION U(NDIM),PAR(*),F(NDIM)
C	
	X1=U(1)
	X2=U(2)
	X3=U(3)
C	
	D=PAR(1)
	ALPHA=PAR(2)
	BETA=PAR(3)
	B=PAR(4)
	S=PAR(5)
C	
	E=DEXP(X3)
	X1C=1-X1
C	
	F(1)=-X1 + D*X1C*E
	F(2)=-X2 + D*E*(X1C - S*X2)
	F(3)=-X3 - BETA*X3 + D*B*E*(X1C + ALPHA*S*X2)
C	
	RETURN
	END

Table 14.2: The equations for demo `abc`, as defined in the equations-file `abc.f`.

```

SUBROUTINE STPNT(NDIM,U,PAR,T)
C -----
C
C      IMPLICIT DOUBLE PRECISION (A-H,O-Z)
C      DIMENSION U(NDIM),PAR(*)
C
C      PAR(1)=0.0
C      PAR(2)=1.0
C      PAR(3)=1.55
C      PAR(4)=8.
C      PAR(5)=0.04
C
C      U(1)=0.
C      U(2)=0.
C      U(3)=0.
C
C      RETURN
C      END

```

Table 14.3: The starting solution for demo **abc**, as defined in the equations-file **abc.f**.

3 1 0 1	NDIM,IPS,IRS,ILP
1 1	NICP,((ICP(I)),I=1,NICP)
15 4 3 1 1 0 0 0	NTST,NCOL,IAD,ISP,ISW,IPLT,NBC,NINT
130 0 0.4 0 25	NMX,RLO,RL1,AO,A1
200 10 2 8 5 3 0	NPR,MXBF,IID,ITMX,ITNW,NWTN,JAC
1e-7 1e-7 1e-4	EPSL,EPSU,EPSS
0.02 0.001 0.1 1	DS,DSMIN,DSMAX,IADS
1	NTHL,((I,THL(I)),I=1,NTHL)
11 0	
0	NTHU,((I,THU(I)),I=1,NTHU)
1	NUZR,((I,UZR(I)),I=1,NUZR)
-1 0.4	

Table 14.4: The constants-file **c.abc.1** for Run 1 (stationary solutions) of demo **abc**.

3 2 2 1	NDIM, IPS, IRS, ILP
2 1 11	NICP, (ICP(I), I=1, NICP)
25 4 3 1 1 0 0 0	NTST, NCOL, IAD, ISP, ISW, IPLT, NBC, NINT
200 0 0.4 0 25	NMX, RLO, RL1, AO, A1
200 10 2 8 5 3 0	NPR, MXBF, IID, ITMX, ITNW, NWTN, JAC
1e-7 1e-7 1e-4	EPSL, EPSU, EPSS
0.02 0.001 0.1 1	DS, DSMIN, DSMAX, IADS
1	NTHL, ((I, THL(I)), I=1, NTHL)
11 0	
0	NTHU, ((I, THU(I)), I=1, NTHU)
1	NUZR, ((I, UZR(I)), I=1, NUZR)
-1 0.25	

Table 14.5: The constants-file `c.abc.2` for Run 2 (periodic orbits) of demo `abc`.

COMMAND	ACTION
<code>mkdir abc</code>	create an empty work directory
<code>cd abc</code>	change directory
<code>@dm abc</code>	copy the demo files to the work directory
<code>@R abc 1</code> <code>@sv abc</code>	compute the stationary solution family with four Hopf bifurcations save output-files as <code>b.abc</code> , <code>s.abc</code> , <code>d.abc</code>
<code>@R abc 2</code> <code>@ap abc</code>	compute a family of periodic solutions from the first Hopf point append the output-files to <code>b.abc</code> , <code>s.abc</code> , <code>d.abc</code>
<code>@R abc 3</code> <code>@ap abc</code>	compute a family of periodic solutions from the second Hopf point append the output-files to <code>b.abc</code> , <code>s.abc</code> , <code>d.abc</code>
<code>@R abc 4</code> <code>@ap abc</code>	compute a family of periodic solutions from the third Hopf point append the output-files to <code>b.abc</code> , <code>s.abc</code> , <code>d.abc</code>
<code>@R abc 5</code> <code>@ap abc</code>	compute a family of periodic solutions from the fourth Hopf point append the output-files to <code>b.abc</code> , <code>s.abc</code> , <code>d.abc</code>

Table 14.6: Unix Commands for running demo `abc`.

```
ld(e='abc',c='abc.1')
run()
sv('abc')
ld(c='abc.2',s='abc')
run()
ap('abc')
ld(c='abc.3',s='abc')
run()
ap('abc')
ld(c='abc.4',s='abc')
run()
ap('abc')
ld(c='abc.5',s='abc')
run()
ap('abc')
```

Table 14.7: Python Commands for running demo abc.

```
ld(e='abc',c='abc.1')
run()
sv('abc')
ld(c='abc.2',s='abc')
data = sl('abc')
for solution in data:
    if solution["Type name"] == "HB":
        ch("IRS", solution["Label"])
        run()
        ap('abc')
```

Table 14.8: Python Program for running demo abc.

14.3 pp2 : A 2D Predator-Prey Model.

This demo illustrates the computation of families of stationary solutions, including bifurcating stationary families, as well as the detection of a Hopf bifurcation. In the second run the family of periodic solutions that emanates from the Hopf bifurcation is computed. This family terminates in a heteroclinic orbit.

The equations, which model a predator-prey system with harvesting, are

$$\begin{aligned} u_1' &= p_2 u_1 (1 - u_1) - u_1 u_2 - p_1 (1 - e^{-p_3 u_1}), \\ u_2' &= -u_2 + p_4 u_1 u_2. \end{aligned} \tag{14.3}$$

Here p_1 is the principal continuation parameter, while $p_2 = p_4 = 3$ and $p_3 = 5$, are fixed. The use of PLAUT is also illustrated. The saved plots are shown in Figure 14.1 and Figure 14.2.

COMMAND	ACTION
<code>mkdir pp2</code>	create an empty work directory
<code>cd pp2</code>	change directory
<code>@dm pp2</code>	copy the demo files to the work directory
<code>@R pp2 1</code>	1st run, using constants-file <code>c.pp2.1</code> ; stationary solutions
<code>@sv pp2</code>	save output-files as <code>b.pp2</code> , <code>s.pp2</code> , <code>d.pp2</code>
<code>@R pp2 2</code>	2nd run, using constants-file <code>c.pp2.2</code> ; periodic solutions
<code>@ap pp2</code>	append output-files to <code>b.pp2</code> , <code>s.pp2</code> , <code>d.pp2</code>

Table 14.9: Commands for running demo pp2.

AUTO-COMMAND	ACTION
@p pp2	run PLAUT to graph the contents of b.pp2 and s.pp2 ;
PLAUT-COMMAND	ACTION
d2	set convenient defaults
ax	select axes
1 3	select real columns 1 and 3 in b.pp2
bd0	plot the bifurcation diagram; $\max u_1$ versus p_1
d1	choose other default settings
bd	get blow-up of current bifurcation diagram
0 1 -0.25 1	enter diagram limits
sav	save plot (see Figure 14.1)
fig.1	upon prompt, enter a new file name, e.g., fig.1
cl	clear the screen
2d	enter 2D mode, for plotting labeled solutions
11 15 19 23	select these labeled orbits in s.pp2
d	default orbit display; u_1 versus time
1 3	select columns 1 and 3 in s.pp2
d	display the orbits; u_2 versus time
2 3	select columns 2 and 3 in s.pp2
d	phase plane display; u_2 versus u_1
sav	save plot (see Figure 14.2)
fig.2	upon prompt, enter a new file name
ex	exit from 2D mode
end	exit from PLAUT

Table 14.10: Plotting commands for demo pp2.

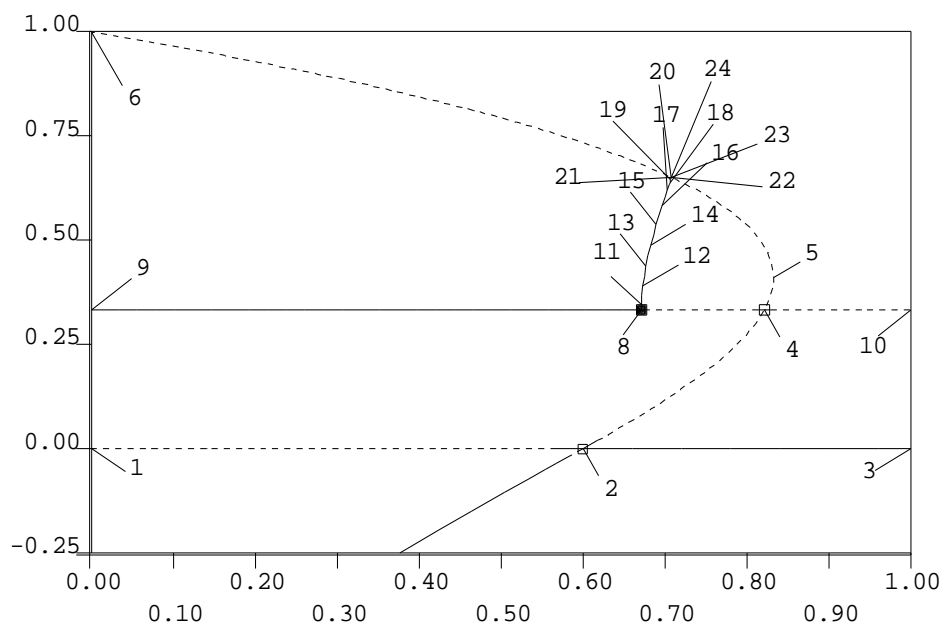


Figure 14.1: The bifurcation diagram of demo pp2.

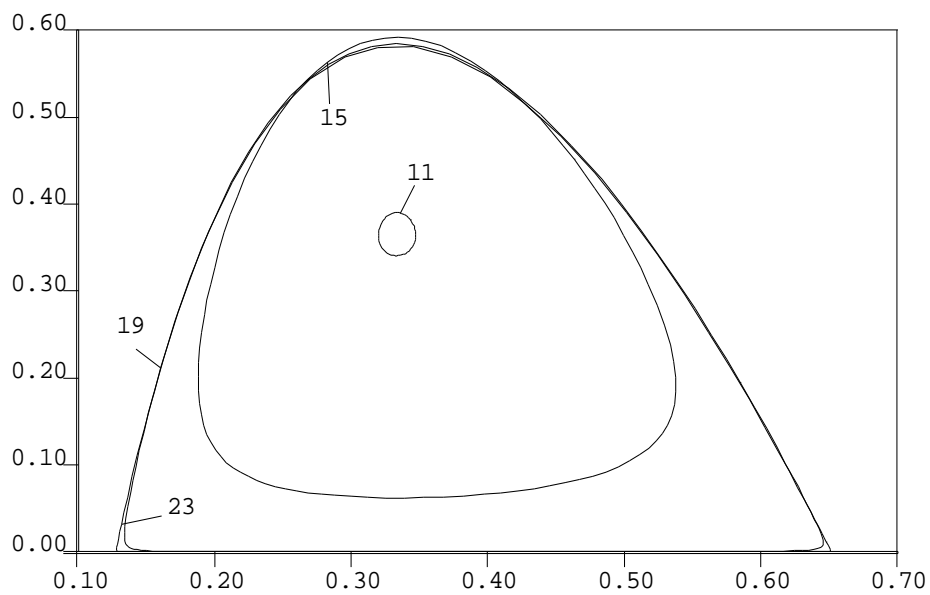


Figure 14.2: The phase plot of solutions 11, 15, 19, and 23 in demo pp2.

14.4 lor : Starting an Orbit from Numerical Data.

This demo illustrates how to start the computation of a family of periodic solutions from numerical data obtained, for example, from an initial value solver. As an illustrative application we consider the Lorenz equations

$$\begin{aligned}u'_1 &= p_3(u_2 - u_1), \\u'_2 &= p_1u_1 - u_2 - u_1u_3, \\u'_3 &= u_1u_2 - p_2u_3.\end{aligned}\tag{14.4}$$

Numerical simulations with a simple initial value solver show the existence of a stable periodic orbit when $p_1 = 280$, $p_2 = 8/3$, $p_3 = 10$. Numerical data representing one complete periodic oscillation are contained in the file `lor.dat`. Each row in `lor.dat` contains four real numbers, namely, the time variable t , u_1 , u_2 and u_3 . The corresponding parameter values are defined in the user-supplied subroutine `stpnt`. The AUTO -command `us('lor')` then converts the data in `lor.dat` to a labeled AUTO solution (with label 1) in a new file `s.dat`. The mesh will be suitably adapted to the solution, using the number of mesh intervals `NTST` and the number of collocation point per mesh interval `NCOL` specified in the constants-file `c.lor`. (Note that the file `s.dat` should be used for restart only. Do not append new output-files to `s.dat`, as the command `us('lor')` only creates `s.dat`, with no corresponding `b.dat`.)

AUTO -COMMAND	ACTION
<code>! mkdir lor</code>	create an empty work directory
<code>cd lor</code>	change directory
<code>demo('lor')</code>	copy the demo files to the work directory
<code>ld('lor')</code>	load the problem definition
<code>us('lor')</code>	convert <code>lor.dat</code> to AUTO format in <code>s.dat</code>
<code>run(c='lor.1',s='dat')</code>	compute a solution family, restart from <code>s.dat</code>
<code>sv('lor')</code>	save output-files as <code>b.lor</code> , <code>s.lor</code> , <code>d.lor</code>
<code>run(c='lor.2',s='lor')</code>	switch branches at a period-doubling detected in the first run. Constants changed : <code>IRS</code> , <code>ISW</code> , <code>NTST</code>
<code>ap('lor')</code>	append the output-files to <code>b.lor</code> , <code>s.lor</code> , <code>d.lor</code>

Table 14.11: Commands for running demo `lor`.

14.5 frc : A Periodically Forced System.

This demo illustrates the computation of periodic solutions to a periodically forced system. In AUTO this can be done by adding a nonlinear oscillator with the desired periodic forcing as one of the solution components. An example of such an oscillator is

$$\begin{aligned}x' &= x + \beta y - x(x^2 + y^2), \\y' &= -\beta x + y - y(x^2 + y^2),\end{aligned}\tag{14.5}$$

which has the asymptotically stable solution $x = \sin(\beta t)$, $y = \cos(\beta t)$. We couple this oscillator to the Fitzhugh-Nagumo equations :

$$\begin{aligned}v' &= (F(v) - w)/\epsilon, \\w' &= v - dw - (b + r \sin(\beta t)),\end{aligned}\tag{14.6}$$

by replacing $\sin(\beta t)$ by x . Above, $F(v) = v(v - a)(1 - v)$ and a, b, ϵ and d are fixed. The first run is a homotopy from $r = 0$, where a solution is known analytically, to $r = 0.2$. Part of the solution family with $r = 0.2$ and varying β is computed in the second run. For detailed results see Alexander, Doedel & Othmer (1990).

AUTO -COMMAND	ACTION
<code>! mkdir frc</code>	create an empty work directory
<code>cd frc</code>	change directory
<code>demo('frc')</code>	copy the demo files to the work directory
<code>ld('frc')</code>	load the problem definition
<code>run(c='frc.1')</code>	homotopy to $r = 0.2$
<code>sv('0')</code>	save output-files as <code>b.0, s.0, d.0</code>
<code>run(c='frc.2',s='0')</code>	compute solution family; restart from <code>s.0</code> . Constants changed : <code>IRS, ICP(1), NTST, NMX, DS, DSMAX</code>
<code>sv('frc')</code>	save output-files as <code>b.frc, s.frc, d.frc</code>

Table 14.12: Commands for running demo `frc`.

14.6 ppp : Continuation of Hopf Bifurcations.

This demo illustrates the continuation of Hopf bifurcations in a 3-dimensional predator prey model (Doedel (1984)). This curve contain branch points, where one locus of Hopf points bifurcates from another locus of Hopf points. The equations are

$$\begin{aligned} u_1' &= u_1(1 - u_1) - p_4 u_1 u_2, \\ u_2' &= -p_2 u_2 + p_4 u_1 u_2 - p_5 u_2 u_3 - p_1(1 - e^{-p_6 u_2}) \\ u_3' &= -p_3 u_3 + p_5 u_2 u_3. \end{aligned} \quad (14.7)$$

Here $p_2 = 1/4$, $p_3 = 1/2$, $p_4 = 3$, $p_5 = 3$, $p_6 = 5$, and p_1 is the free parameter. In the continuation of Hopf points the parameter p_4 is also free.

AUTO -COMMAND	ACTION
<code>! mkdir ppp</code> <code>cd ppp</code> <code>demo('ppp')</code>	create an empty work directory change directory copy the demo files to the work directory
<code>ld('ppp')</code> <code>run(c='ppp.1')</code> <code>sv('ppp')</code>	load the problem definition compute stationary solutions; detect Hopf bifurcations save output-files as b.ppp, s.ppp, d.ppp
<code>run(c='ppp.2',s='ppp')</code> <code>ap('ppp')</code>	compute a family of periodic solutions. Constants changed : IPS, IRS, ICP append the output-files to b.ppp, s.ppp, d.ppp
<code>run(c='ppp.3',s='ppp')</code> <code>sv('hb')</code>	compute Hopf bifurcation curves save the output-files as b.hb, s.hb, d.hb

Table 14.13: Commands for running demo ppp.

14.7 plp : Fold Continuation for Periodic Solutions.

This demo, which corresponds to computations in Doedel, Keller & Kernévez (1991*a*), shows how one can continue a fold on a family of periodic solution in two parameters. The calculation of a locus of Hopf bifurcations is also included. The equations, that model a one-compartment activator-inhibitor system (Kernévez (1980)), are given by

$$\begin{aligned} s' &= (s_0 - s) - \rho R(s, a), \\ a' &= \alpha(a_0 - a) - \rho R(s, a), \end{aligned} \quad (14.8)$$

where

$$R(s, a) = \frac{sa}{1 + s + \kappa s^2}, \quad \kappa > 0.$$

The free parameter is ρ . In the fold continuation s_0 is also free.

AUTO -COMMAND	ACTION
! mkdir plp cd plp demo('plp')	create an empty work directory change directory copy the demo files to the work directory
ld('plp') run(c='plp.1') sv('plp')	load the problem definition 1st run; compute a stationary solution family and locate HBs save output-files as b.plp, s.plp, d.plp
run(c='plp.2',s='plp') ap('plp')	compute a family of periodic solutions and locate a fold. Constants changed : IPS, IRS, NMX append output-files to b.plp, s.plp, d.plp
run(c='plp.3',s='plp') sv('2p')	Compute a locus of Hopf bifurcation points. Constants changed : IPS, ICP, ISW, NMX, RL1 save output-files as b.2p, s.2p, d.2p
run(c='plp.4',s='plp') sv('tmp')	generate starting data for the fold contin- uation. Constants changed : IPS, IRS, ICP, NMX save output-files as b.tmp, s.tmp, d.tmp
run(c='plp.5',s='tmp') ap('2p')	fold continuation; restart data from s.tmp. Constants changed : IRS, NUZR append output-files to b.2p, s.2p, d.2p
run(c='plp.6',s='2p') sv('iso')	compute an isola of periodic solutions; restart data from s.2p. Constants changed : IRS, ISW, NMX, NUZR save output-files as b.iso, s.iso, d.iso

Table 14.14: Commands for running demo plp.

14.8 pp3 : Periodic Families and Loci of Hopf Points.

This demo illustrates the computation of stationary solution families that contain Hopf bifurcations, and the computation of the emanating families of periodic solutions. In this example the periodic solution families intersect at a secondary bifurcation point (a branch point). It also shows how to compute a locus of Hopf bifurcation points in two parameters. (In this example the locus contains branch points, which lead to another locus!)

The equations, which model a 3D predator-prey system with harvesting (Doedel (1984)), are

$$\begin{aligned} u_1' &= u_1(1 - u_1) - p_4 u_1 u_2, \\ u_2' &= -p_2 u_2 + p_4 u_1 u_2 - p_5 u_2 u_3 - p_1(1 - e^{-p_6 u_2}) \\ u_3' &= -p_3 u_3 + p_5 u_2 u_3. \end{aligned} \tag{14.9}$$

The free parameter is p_1 , while the other parameters are fixed, namely $p_2 = 0.25$, $p_3 = 0.5$, $p_4 = 4$, $p_5 = 3$, and $p_6 = 5$. However, both p_1 and p_4 are free in the computation of loci of Hopf points.

COMMAND	ACTION
<code>mkdir pp3</code>	create an empty work directory
<code>cd pp3</code>	change directory
<code>@dm pp3</code>	copy the demo files to the work directory
<code>@R pp3 1</code> <code>@sv pp3</code>	1st run; stationary solutions with 4 Hopf bifurcations save output-files as <code>b.pp3</code> , <code>s.pp3</code> , <code>d.pp3</code>
<code>@R pp3 2</code> <code>@ap pp3</code>	compute a family of periodic solutions append output-files to <code>b.pp3</code> , <code>s.pp3</code> , <code>d.pp3</code>
<code>@R pp3 3</code> <code>@ap pp3</code>	compute another family of periodic solutions append output-files to <code>b.pp3</code> , <code>s.pp3</code> , <code>d.pp3</code>
<code>@R pp3 4</code> <code>@rl</code> <code>@sv hb</code>	compute loci of Hopf points relabel the labeled solutions from this run save the output-files as <code>b.hb</code> , <code>s.hb</code> , <code>d.pp3</code>

Table 14.15: Commands for running demo pp3.

AUTO-COMMAND	ACTION
@p pp3	run PLAUT to graph the contents of b.pp3 and s.pp3 ;
PLAUT-COMMAND	ACTION
d2	set convenient defaults
ax	select axes
1 3	select real columns 1 and 3 in b.pp3
bd0	plot the bifurcation diagram; $\max u_1$ versus p_1
bd	get blow-up of current bifurcation diagram
0 0.6 0 1.2	enter diagram limits
d1	choose other default settings (with labels)
bd	another blow-up of the bifurcation diagram
0 0.6 0 0.75	enter diagram limits
d2	set defaults
2d	enter 2D mode, for plotting labeled solutions
13 15 17 19 21 23 25 27	select these orbits from s.pp3
d	default orbit display; u_1 versus time
2 3	select columns 2 and 3 in s.pp3
d	display the orbits; u_2 versus u_1
2d	enter 2D mode, for plotting labeled solutions
28 30 32 34 36 38 40 42 44	select these orbits
d	default orbit display; u_1 versus time
2 3	select columns 2 and 3 in s.pp3
d	phase plane display; u_2 versus u_1
2 4	select columns 2 and 4 in s.pp3
d	phase plane display; u_3 versus u_1
ex	exit from 2D mode
end	exit from PLAUT

Table 14.16: Plotting commands for demo pp3.

AUTO-COMMAND	ACTION
@p hb	run PLAUT to graph the contents of b.hb and s.hb ;
PLAUT-COMMAND	ACTION
d0	set defaults
ax	select axes
1 6	select real columns 1 and 6 in b.hb
bd0	plot the bifurcation diagram; p_4 versus p_1
end	exit from PLAUT

Table 14.17: Plotting the Hopf loci for demo pp3.

14.9 tor : Detection of Torus Bifurcations.

This demo uses a model in Freire, Rodríguez-Luis, Gamero & Ponce (1993) to illustrate the detection of a torus bifurcation. It also illustrates branch switching at a secondary periodic bifurcation with double Floquet multiplier at $z = 1$. The computational results also include folds, homoclinic orbits, and period-doubling bifurcations. Their continuation is not illustrated here; see instead the demos `plp`, `pp2`, and `pp3`, respectively. The equations are

$$\begin{aligned}x'(t) &= [- (\beta + \nu)x + \beta y - a_3 x^3 + b_3 (y - x)^3] / r, \\y'(t) &= \beta x - (\beta + \gamma)y - z - b_3 (y - x)^3, \\z'(t) &= y,\end{aligned}\tag{14.10}$$

where $\gamma = -0.6$, $r = 0.6$, $a_3 = 0.328578$, and $b_3 = 0.933578$. Initially $\nu = -0.9$ and $\beta = 0.5$.

AUTO -COMMAND	ACTION
<code>! mkdir tor</code> <code>cd tor</code> <code>demo('tor')</code>	create an empty work directory change directory copy the demo files to the work directory
<code>ld('tor')</code> <code>run(c='tor.1')</code> <code>sv('1')</code>	load the problem definition 1st run; compute a stationary solution family with Hopf bifurcation save output-files as <code>b.1</code> , <code>s.1</code> , <code>d.1</code>
<code>run(c='tor.2',s='1')</code> <code>ap('1')</code>	compute a family of periodic solutions; restart from <code>s.1</code> . Constants changed : <code>IPS</code> , <code>IRS</code> append output-files to <code>b.1</code> , <code>s.1</code> , <code>d.1</code>
<code>run(c='tor.3',s='1')</code> <code>ap('1')</code>	compute a bifurcating family of periodic solutions; restart from <code>s.1</code> . Constants changed : <code>IRS</code> , <code>ISW</code> , <code>NMX</code> append output-files to <code>b.1</code> , <code>s.1</code> , <code>d.1</code>

Table 14.18: Commands for running demo `tor`.

14.10 pen : Rotations of Coupled Pendula.

This demo illustrates the computation of rotations, i.e., solutions that are periodic, modulo a phase gain of an even multiple of π . AUTO checks the starting data for components with such a phase gain and, if present, it will automatically adjust the computations accordingly. The model equations, a system of two coupled pendula, (Doedel, Aronson & Othmer (1991)), are given by

$$\begin{aligned}\phi_1'' + \epsilon\phi_1' + \sin \phi_1 &= I + \gamma(\phi_2 - \phi_1), \\ \phi_2'' + \epsilon\phi_2' + \sin \phi_2 &= I + \gamma(\phi_1 - \phi_2),\end{aligned}\tag{14.11}$$

or, in equivalent first order form,

$$\begin{aligned}\phi_1' &= \psi_1, \\ \phi_2' &= \psi_2, \\ \psi_1' &= -\epsilon\psi_1 - \sin \phi_1 + I + \gamma(\phi_2 - \phi_1), \\ \psi_2' &= -\epsilon\psi_2 - \sin \phi_2 + I + \gamma(\phi_1 - \phi_2).\end{aligned}\tag{14.12}$$

Throughout $\gamma = 0.175$. Initially, $\epsilon = 0.1$ and $I = 0.4$.

Numerical data representing one complete rotation are contained in the file **pen.dat**. Each row in **pen.dat** contains five real numbers, namely, the time variable t , ϕ_1 , ϕ_2 , ψ_1 and ψ_2 . The corresponding parameter values are defined in the user-supplied subroutine **STPNT**.

Actually, in this example, a scaled time variable t is given in **pen.dat**. For this reason the period (`PAR(11)`) is also set in **STPNT**. Normally AUTO would automatically set the period according to the data in **pen.dat**.

The AUTO -command `us('pen')` converts the data in **pen.dat** to a labeled AUTO solution (with label 1) in a new file **s.dat**. The mesh will be suitably adapted to the solution, using the number of mesh intervals `NTST` and the number of collocation point per mesh interval `NCOL` specified in the constants-file **c.pen**. (Note that the file **s.dat** should be used for restart only. Do not append new output-files to **s.dat**, as the command `us('pen')` only creates **s.dat**, with no corresponding **b.dat**.)

The first run, with I as free problem parameter, starts from the converted solution with label 1 in **pen.dat**. A period-doubling bifurcation is located, and the period-doubled family is computed in the second run. Two branch points are located, and the bifurcating families are traced out in the third and fourth run, respectively. The fifth run generates starting data for the subsequent computation of a locus of period-doubling bifurcations. The actual computation is done in the sixth run, with ϵ and I as free problem parameters.

AUTO -COMMAND	ACTION
! mkdir pen cd pen demo('pen')	create an empty work directory change directory copy the demo files to the work directory
ld('pen') us('pen')	load the problem definition convert pen.dat to AUTO format in s.dat
run(c='pen.1',s='dat') sv('pen')	locate a period doubling bifurcation; restart from s.dat save output-files as b.pen, s.pen, d.pen
run(c='pen.2',s='pen') ap('pen')	a family of period-doubled (and out-of-phase) rotations. Constants changed : IPS, NTST, ISW, NMX append output-files to b.pen, s.pen, d.pen
run(c='pen.3',s='pen') ap('pen')	a secondary bifurcating family (without bifurcation detection). Constants changed : IRS, ISP append output-files to b.pen, s.pen, d.pen
run(c='pen.4',s='pen') ap('pen')	another secondary bifurcating family (without bifurcation detection). Constants changed : IRS append output-files to b.pen, s.pen, d.pen
run(c='pen.5',s='pen') sv('t')	generate starting data for period doubling continuation. Constants changed : IRS, ICP, ICP, ISW, NMX save output-files as b.t, s.t, d.t
run(c='pen.6',s='t') sv('pd')	compute a locus of period doubling bifurcations; restart from s.t. Constants changed : IRS save output-files as b.pd, s.pd, d.pd

Table 14.19: Commands for running demo pen.

14.11 chu : A Non-Smooth System (Chua's Circuit).

Chua's circuit is one of the simplest electronic devices to exhibit complex behavior. For related calculations see Khibnik, Roose & Chua (1993). The equations modeling the circuit are

$$\begin{aligned} u_1' &= \alpha [u_2 - h(u_1)] , \\ u_2' &= u_1 - u_2 + u_3 , \\ u_3' &= -\beta u_2 , \end{aligned} \tag{14.13}$$

where

$$h(x) = a_1 x + \frac{1}{2} (a_0 - a_1) \{ |x + 1| - |x - 1| \} ,$$

and where we take $\beta = 14.3$, $a_0 = -1/7$, $a_1 = 2/7$.

Note that $h(x)$ is not a smooth function, and hence the solution to the equations may have non-smooth derivatives. However, for the orthogonal collocation method to attain its optimal accuracy, it is necessary that the solution be sufficiently smooth. Moreover, the adaptive mesh selection strategy will fail if the solution or one of its lower order derivatives has discontinuities. For these reasons we use the smooth approximation

$$|x| \approx \frac{2x}{\pi} \arctan(Kx),$$

which get better as K increases. In the numerical calculations below we use $K = 10$. The free parameter is α .

AUTO -COMMAND	ACTION
<code>! mkdir chu</code>	create an empty work directory
<code>cd chu</code>	change directory
<code>demo('chu')</code>	copy the demo files to the work directory
<code>ld('chu')</code>	load the problem definition
<code>run(c='chu.1')</code>	1st run; stationary solutions
<code>sv('chu')</code>	save output-files as <code>b.chu</code> , <code>s.chu</code> , <code>d.chu</code>
<code>run(c='chu.2',s='chu')</code>	2nd run; periodic solutions, with detection of period-doubling. constants changed : IPS, IRS, ICP, ICP
<code>ap('chu')</code>	append the output-files to <code>b.chu</code> , <code>s.chu</code> , <code>d.chu</code>

Table 14.20: Commands for running demo chu.

14.12 phs : Effect of the Phase Condition.

This demo illustrates the effect of the phase condition on the computation of periodic solutions. We consider the differential equation

$$\begin{aligned} u_1' &= \lambda u_1 - u_2, \\ u_2' &= u_1(1 - u_1). \end{aligned} \tag{14.14}$$

This equation has a Hopf bifurcation from the trivial solution at $\lambda = 0$. The bifurcating family of periodic solutions is vertical and along it the period increases monotonically. The family terminates in a homoclinic orbit containing the saddle point $(u_1, u_2) = (1, 0)$. Graphical inspection of the computed periodic orbits, for example u_1 versus the scaled time variable t , shows how the phase condition has the effect of keeping the “peak” in the solution in the same location.

AUTO -COMMAND	ACTION
<code>! mkdir phs</code>	create an empty work directory
<code>cd phs</code>	change directory
<code>demo('phs')</code>	copy the demo files to the work directory
<code>ld('phs')</code>	load the problem definition
<code>run(c='phs.1')</code>	detect Hopf bifurcation
<code>sv('phs')</code>	save output-files as <code>b.phs</code> , <code>s.phs</code> , <code>d.phs</code>
<code>run(c='phs.2', s='phs')</code>	compute periodic solutions. Constants changed : <code>IRS</code> , <code>IPS</code> , <code>NPR</code>
<code>ap('phs')</code>	append output-files to <code>b.phs</code> , <code>s.phs</code> , <code>d.phs</code>

Table 14.21: Commands for running demo `phs`.

14.13 ivp : Time Integration with Euler's Method.

This demo uses Euler's method to locate a stationary solution of the following predator-prey system with harvesting :

$$\begin{aligned} u_1' &= p_2 u_1 (1 - u_1) - u_1 u_2 - p_1 (1 - e^{-p_3 u_1}), \\ u_2' &= -u_2 + p_4 u_1 u_2, \end{aligned} \quad (14.15)$$

where all problem parameters have a fixed value. The equations are the same as those in demo `pp2`. The continuation parameter is the independent time variable, namely `PAR(14)`.

Note that Euler time integration is only first order accurate, so that the time step must be sufficiently small to ensure correct results. Indeed, this option has been added only as a convenience, and should generally be used only to locate stationary states. Note that the AUTO -constants `DS`, `DSMIN`, and `DSMAX` control the step size in the space consisting of time, here `PAR(14)`, and the state vector, here (u_1, u_2) .

AUTO -COMMAND	ACTION
<code>! mkdir ivp</code>	create an empty work directory
<code>cd ivp</code>	change directory
<code>demo('ivp')</code>	copy the demo files to the work directory
<code>ld('ivp')</code>	load the problem definition
<code>run(c='ivp.1')</code>	time integration
<code>sv('ivp')</code>	save output-files as <code>b.ivp</code> , <code>s.ivp</code> , <code>d.ivp</code>

Table 14.22: Commands for running demo `ivp`.

Chapter 15

AUTO Demos : BVP.

15.1 exp : Bratu's Equation.

This demo illustrates the computation of a solution family to the boundary value problem

$$\begin{aligned} u_1' &= u_2, \\ u_2' &= -p_1 e^{u_1}, \end{aligned} \tag{15.1}$$

with boundary conditions $u_1(0) = 0$, $u_1(1) = 0$. This equation is also considered in Doedel, Keller & Kernévez (1991a).

AUTO -COMMAND	ACTION
! mkdir exp cd exp demo('exp')	create an empty work directory change directory copy the demo files to the work directory
run(c='exp.1') sv('exp')	1st run; compute solution family containing fold save output-files as b.exp , s.exp , d.exp
run(c='exp.2',s='exp') ap('exp')	2nd run; restart at a labeled solution, using increased accuracy. Constants changed : IRS, NTST, A1, DSMAX vspace0.2cm append output-files to b.exp , s.exp , d.exp

Table 15.1: Commands for running demo exp.

15.2 int : Boundary and Integral Constraints.

This demo illustrates the computation of a solution family to the equation

$$\begin{aligned} u_1' &= u_2, \\ u_2' &= -p_1 e^{u_1}, \end{aligned} \tag{15.2}$$

with a non-separated boundary condition and an integral constraint:

$$u_1(0) - u_1(1) - p_2 = 0, \quad \int_0^1 u(t)dt - p_3 = 0.$$

The solution family contains a fold, which, in the second run, is continued in two equation parameters.

AUTO -COMMAND	ACTION
! mkdir int cd int demo('int')	create an empty work directory change directory copy the demo files to the work directory
run(c='int.1') sv('int')	1st run; detection of a fold save output-files as b.int, s.int, d.int
run(c='int.2',s='int') sv('t')	2nd run; generate starting data for a curve of folds. Constants changed pace0.2cm save the output-files as b.t, s.t, d.t
run(c='int.3',s='t') sv('lp')	3rd run; compute a curve of folds; restart from s.t. Constants changed : IRS vs- pace0.2cm save the output-files as b.lp, s.lp, d.lp

Table 15.2: Commands for running demo int.

15.3 bvp : A Nonlinear ODE Eigenvalue Problem.

This demo illustrates the location of eigenvalues of a nonlinear ODE boundary value problem as bifurcations from the trivial solution family. The family of solutions that bifurcates at the first eigenvalue is computed in both directions. The equations are

$$\begin{aligned} u_1' &= u_2, \\ u_2' &= -(p_1\pi)^2 u_1 + u_1^2, \end{aligned} \tag{15.3}$$

with boundary conditions $u_1(0) = 0$, $u_1(1) = 0$.

AUTO -COMMAND	ACTION
! mkdir bvp cd bvp demo('bvp')	create an empty work directory change directory copy the demo files to the work directory
run(c='bvp.1') sv('bvp')	compute the trivial solution family and locate eigenvalues save output-files as b.bvp , s.bvp , d.bvp
run(c='bvp.2',s='bvp') ap('bvp')	compute the first bifurcating family. Constants changed : IRS, ISW, NPR, DSMAX append output-files to b.bvp , s.bvp , d.bvp
run(c='bvp.3',s='bvp') ap('bvp')	compute the first bifurcating family in opposite direction. Constants changed : DS append output-files to b.bvp , s.bvp , d.bvp

Table 15.3: Commands for running demo **bvp**.

15.4 lin : A Linear ODE Eigenvalue Problem.

This demo illustrates the location of eigenvalues of a linear ODE boundary value problem as bifurcations from the trivial solution family. By means of branch switching an eigenfunction is computed, as is illustrated for the first eigenvalue. This eigenvalue is then continued in two parameters by fixing the L_2 -norm of the first solution component. The eigenvalue problem is given by the equations

$$\begin{aligned} u_1' &= u_2, \\ u_2' &= (p_1\pi)^2 u_1, \end{aligned} \tag{15.4}$$

with boundary conditions $u_1(0) - p_2 = 0$ and $u_1(1) = 0$. We add the integral constraint

$$\int_0^1 u_1(t)^2 dt - p_3 = 0.$$

Then p_3 is simply the L_2 -norm of the first solution component. In the first two runs p_2 is fixed, while p_1 and p_3 are free. In the third run p_3 is fixed, while p_1 and p_2 are free.

AUTO -COMMAND	ACTION
<code>! mkdir lin</code> <code>cd lin</code> <code>demo('lin')</code>	create an empty work directory change directory copy the demo files to the work directory
<code>run(c='lin.1')</code> <code>sv('lin')</code>	1st run; compute the trivial solution family and locate eigenvalues save output-files as <code>b.lin</code> , <code>s.lin</code> , <code>d.lin</code>
<code>run(c='lin.2',s='lin')</code> <code>ap('lin')</code>	2nd run; compute a few steps along the bifurcating family. Constants changed : <code>IRS</code> , <code>ISW</code> , <code>DSMAX</code> append output-files to <code>b.lin</code> , <code>s.lin</code> , <code>d.lin</code>
<code>run(c='lin.3',s='lin')</code> <code>sv('2p')</code>	3rd run; compute a two-parameter curve of eigenvalues. Constants changed : <code>IRS</code> , <code>ISW</code> , <code>ICP(2)</code> save the output-files as <code>b.2p</code> , <code>s.2p</code> , <code>d.2p</code>

Table 15.4: Commands for running demo `lin`.

15.5 non : A Non-Autonomous BVP.

This demo illustrates the continuation of solutions to the non-autonomous boundary value problem

$$\begin{aligned} u_1' &= u_2, \\ u_2' &= -p_1 e^{x^3 u_1}, \end{aligned} \tag{15.5}$$

with boundary conditions $u_1(0) = 0$, $u_1(1) = 0$. Here x is the independent variable. This system is first converted to the following equivalent autonomous system :

$$\begin{aligned} u_1' &= u_2, \\ u_2' &= -p_1 e^{u_3^3 u_1}, \\ u_3' &= 1, \end{aligned} \tag{15.6}$$

with boundary conditions $u_1(0) = 0$, $u_1(1) = 0$, $u_3(0) = 0$. (For a periodically forced system see demo `frc`).

AUTO -COMMAND	ACTION
<code>! mkdir non</code>	create an empty work directory
<code>cd non</code>	change directory
<code>demo('non')</code>	copy the demo files to the work directory
<code>run(c='non.1')</code>	compute the solution family
<code>sv('non')</code>	save output-files as <code>b.non</code> , <code>s.non</code> , <code>d.non</code>

Table 15.5: Commands for running demo `non`.

15.6 kar : The Von Karman Swirling Flows.

The steady axi-symmetric flow of a viscous incompressible fluid above an infinite rotating disk is modeled by the following ODE boundary value problem (Equation (11) in Lentini & Keller (1980)) :

$$\begin{aligned} u_1' &= Tu_2, \\ u_2' &= Tu_3, \\ u_3' &= T[-2\gamma u_4 + u_2^2 - 2u_1u_3 - u_4^2], \\ u_4' &= Tu_5, \\ u_5' &= T[2\gamma u_2 + 2u_2u_4 - 2u_1u_5], \end{aligned} \quad (15.7)$$

with left boundary conditions

$$u_1(0) = 0, \quad u_2(0) = 0, \quad u_4(0) = 1 - \gamma,$$

and (asymptotic) right boundary conditions

$$\begin{aligned} [f_\infty + a(f_\infty, \gamma)] u_2(1) + u_3(1) - \gamma \frac{u_4(1)}{a(f_\infty, \gamma)} &= 0, \\ a(f_\infty, \gamma) \frac{b^2(f_\infty, \gamma)}{\gamma} u_2(1) + [f_\infty + a(f_\infty, \gamma)] u_4(1) + u_5(1) &= 0, \\ u_1(1) &= f_\infty, \end{aligned} \quad (15.8)$$

where

$$\begin{aligned} a(f_\infty, \gamma) &= \frac{1}{\sqrt{2}} [(f_\infty^4 + 4\gamma^2)^{1/2} + f_\infty^2]^{1/2}, \\ b(f_\infty, \gamma) &= \frac{1}{\sqrt{2}} [(f_\infty^4 + 4\gamma^2)^{1/2} - f_\infty^2]^{1/2}. \end{aligned} \quad (15.9)$$

Note that there are five differential equations and six boundary conditions. Correspondingly, there are two free parameters in the computation of a solution family, namely γ and f_∞ . The “period” T is fixed; $T = 500$. The starting solution is $u_i = 0$, $i = 1, \dots, 5$, at $\gamma = 1$, $f_\infty = 0$.

AUTO -COMMAND	ACTION
<code>! mkdir kar</code>	create an empty work directory
<code>cd kar</code>	change directory
<code>demo('kar')</code>	copy the demo files to the work directory
<code>run(c='kar.1')</code>	computation of the solution family
<code>sv('kar')</code>	save output-files as <code>b.kar</code> , <code>s.kar</code> , <code>d.kar</code>

Table 15.6: Commands for running demo `kar`.

15.7 spb : A Singularly-Perturbed BVP.

This demo illustrates the use of continuation to compute solutions to the singularly perturbed boundary value problem

$$\begin{aligned} u_1' &= u_2, \\ u_2' &= \frac{\lambda}{\epsilon} (u_1 u_2 (u_1^2 - 1) + u_1), \end{aligned} \quad (15.10)$$

with boundary conditions $u_1(0) = 3/2$, $u_1(1) = \gamma$. The parameter λ has been introduced into the equations in order to allow a homotopy from a simple equation with known exact solution to the actual equation. This is done in the first run. In the second run ϵ is decreased by continuation. In the third run ϵ is fixed at $\epsilon = .001$ and the solution is continued in γ . This run takes more than 1500 continuation steps. For a detailed analysis of the solution behavior see Lorenz (1982).

AUTO -COMMAND	ACTION
<code>! mkdir spb</code> <code>cd spb</code> <code>demo('spb')</code>	create an empty work directory change directory copy the demo files to the work directory
<code>run(c='spb.1')</code> <code>sv('1')</code>	1st run; homotopy from $\lambda = 0$ to $\lambda = 1$ save output-files as <code>b.1</code> , <code>s.1</code> , <code>d.1</code>
<code>run(c='spb.2',s='1')</code> <code>sv('2')</code>	2nd run; let ϵ tend to zero; restart from <code>s.1</code> . constants changed : <code>IRS</code> , <code>ICP(1)</code> , <code>NTST</code> , <code>DS</code> save the output-files as <code>b.2</code> , <code>s.2</code> , <code>d.2</code>
<code>run(c='spb.3',s='2')</code> <code>sv('3')</code>	3rd run; continuation in γ ; $\epsilon = 0.001$; restart from <code>s.2</code> . Constants changed : <code>IRS</code> , <code>ICP(1)</code> , <code>RL0</code> , <code>ITNW</code> , <code>EPSL</code> , <code>EPSU</code> , <code>NUZR</code> save the output-files as <code>b.3</code> , <code>s.3</code> , <code>d.3</code>

Table 15.7: Commands for running demo `spb`.

15.8 `ezp` : Complex Bifurcation in a BVP.

This demo illustrates the computation of a solution family to the the complex boundary value problem

$$\begin{aligned} u_1' &= u_2, \\ u_2' &= -p_1 e^{u_1}, \end{aligned} \tag{15.11}$$

with boundary conditions $u_1(0) = 0$, $u_1(1) = 0$. Here u_1 and u_2 are allowed to be complex, while the parameter p_1 can only take real values. In the real case, this is Bratu's equation, whose solution family contains a fold; see the demo `exp`. It is known (Henderson & Keller (1990)) that a simple quadratic fold gives rise to a pitch fork bifurcation in the complex equation. This bifurcation is located in the first computation below. In the second and third run, both legs of the bifurcating solution family are computed. On it, both solution components u_1 and u_2 have nontrivial imaginary part.

AUTO -COMMAND	ACTION
<code>! mkdir ezp</code> <code>cd ezp</code> <code>demo('ezp')</code>	create an empty work directory change directory copy the demo files to the work directory
<code>run(c='ezp.1')</code> <code>sv('ezp')</code>	1st run; compute solution family containing fold save output-files as <code>p.ezp</code> , <code>s.ezp</code> , <code>d.ezp</code>
<code>run(c='ezp.2',s='ezp')</code> <code>ap('ezp')</code>	2nd run; compute bifurcating complex solution family. Constants changed : <code>IRS</code> , <code>ISW</code> append output-files to <code>p.ezp</code> , <code>s.ezp</code> , <code>d.ezp</code>
<code>run(c='ezp.3',s='ezp')</code> <code>ap('ezp')</code>	3rd run; compute 2nd leg of bifurcating family. constant changed : <code>DS</code> append output-files to <code>p.ezp</code> , <code>s.ezp</code> , <code>d.ezp</code>

Table 15.8: Commands for running demo `ezp`.

Chapter 16

AUTO Demos : Parabolic PDEs.

16.1 pd1 : Stationary States (1D Problem).

This demo uses Euler's method to locate a stationary solution of a nonlinear parabolic PDE, followed by continuation of this stationary state in a free problem parameter. The equation is

$$\frac{\partial u}{\partial t} = D \frac{\partial^2 u}{\partial x^2} + p_1 u (1 - u),$$

on the space interval $[0, L]$, where $L = \text{PAR}(11) = 10$ is fixed throughout, as is the diffusion constant $D = \text{PAR}(15) = 0.1$. The boundary conditions are $u(0) = u(L) = 0$ for all time.

In the first run the continuation parameter is the independent time variable, namely $\text{PAR}(14)$, while $p_1 = 1$ is fixed. The AUTO -constants DS , DSMIN , and DSMAX then control the step size in space-time, here consisting of $\text{PAR}(14)$ and $u(x)$. Initial data are $u(x) = \sin(\pi x/L)$ at time zero. Note that in the subroutine **STPNT** the initial data must be scaled to the unit interval, and that the scaled derivative must also be provided; see the equations-file **pv1.f**. In the second run the continuation parameter is p_1 .

Euler time integration is only first order accurate, so that the time step must be sufficiently small to ensure correct results. Indeed, this option has been added only as a convenience, and should generally be used only to locate stationary states.

AUTO -COMMAND	ACTION
<code>! mkdir pd1</code>	create an empty work directory
<code>cd pd1</code>	change directory
<code>demo('pd1')</code>	copy the demo files to the work directory
<code>run(c='pd1.1')</code> <code>sv('1')</code>	time integration towards stationary state save output-files as b.1, s.1, d.1
<code>run(c='pd1.2', s='1')</code> <code>sv('2')</code>	continuation of stationary states; read restart data from s.1 . constants changed : IPS, IRS, ICP , etc. save output-files as b.2, s.2, d.2

Table 16.1: Commands for running demo **pd1**.

16.2 pd2 : Stationary States (2D Problem).

This demo uses Euler's method to locate a stationary solution of a nonlinear parabolic PDE, followed by continuation of this stationary state in a free problem parameter. The equations are

$$\begin{aligned}\partial u_1/\partial t &= D_1 \partial^2 u_1/\partial x^2 + p_1 u (1 - u) - u_1 u_2, \\ \partial u_2/\partial t &= D_2 \partial^2 u_2/\partial x^2 - u_2 + u_1 u_2,\end{aligned}\tag{16.1}$$

on the space interval $[0, L]$, where $L = \text{PAR}(11) = 1$ is fixed throughout, as are the diffusion constants $D_1 = \text{PAR}(15) = 1$ and $D_2 = \text{PAR}(16) = 1$. The boundary conditions are $u_1(0) = u_1(L) = 0$ and $u_2(0) = u_2(L) = 1$, for all time.

In the first run the continuation parameter is the independent time variable, namely $\text{PAR}(14)$, while $p_1 = 12$ is fixed. The AUTO -constants DS , DSMIN , and DSMAX then control the step size in space-time, here consisting of $\text{PAR}(14)$ and $(u_1(x), u_2(x))$. Initial data at time zero are $u_1(x) = \sin(\pi x/L)$ and $u_2(x) = 1$. Note that in the subroutine **STPNT** the initial data must be scaled to the unit interval, and that the scaled derivatives must also be provided; see the equations-file **pv2.f**. In the second run the continuation parameter is p_1 . A branch point is located during this run.

Euler time integration is only first order accurate, so that the time step must be sufficiently small to ensure correct results. Indeed, this option has been added only as a convenience, and should generally be used only to locate stationary states.

AUTO -COMMAND	ACTION
<code>! mkdir pd2</code>	create an empty work directory
<code>cd pd2</code>	change directory
<code>demo('pd2')</code>	copy the demo files to the work directory
<code>run(c='pd2.1')</code> <code>sv('1')</code>	time integration towards stationary state save output-files as b.1, s.1, d.1
<code>run(c='pd2.2',s='1')</code> <code>sv('2')</code>	continuation of stationary states; read restart data from s.1 . constants changed : IPS, IRS, ICP , etc. save output-files as b.2, s.2, d.2

Table 16.2: Commands for running demo **pd2**.

16.3 wav : Periodic Waves.

This demo illustrates the computation of various periodic wave solutions to a system of coupled parabolic partial differential equations on the spatial interval $[0, 1]$. The equations, that model an enzyme catalyzed reaction (Doedel & Kernévez (1986b)) are :

$$\begin{aligned}\partial u_1/\partial t &= \partial^2 u_1/\partial x^2 - p_1 [p_4 R(u_1, u_2) - (p_2 - u_1)], \\ \partial u_2/\partial t &= \beta \partial^2 u_2/\partial x^2 - p_1 [p_4 R(u_1, u_2) - p_7(p_3 - u_2)].\end{aligned}\tag{16.2}$$

All equation parameters, except p_3 , are fixed throughout.

AUTO -COMMAND	ACTION
! mkdir wav cd wav demo('wav')	create an empty work directory change directory copy the demo files to the work directory
run(c='wav.1') sv('ode')	1st run; stationary solutions of the system without diffusion save output-files as b.ode, s.ode, d.ode
cp c.wav.2 c.wav run(c='wav.2',s='wav') sv('wav')	constants changed : IPS 2nd run; detect bifurcations to wave train solutions. Constants changed : IPS save output-files as b.wav, s.wav, d.wav
run(c='wav.3',s='wav') ap('wav')	3rd run; wave train solutions of fixed wave speed. Constants changed : IRS, IPS, NUZR, ILP append output-files to b.wav, s.wav, d.wav
run(c='wav.4',s='wav') sv('rng')	4th run; wave train solutions of fixed wave length. Constants changed : IRS, IPS, NMX, ICP, NUZR save output-files as b.rng, s.rng, d.rng
run(c='wav.5',s='wav') sv('tim')	5th run; time evolution computation. Con- stants changed : IPS, NMX, NPR, ICP save output-files as b.tim, s.tim, d.tim

Table 16.3: Commands for running demo wav.

16.4 brc : Chebyshev Collocation in Space.

This demo illustrates the computation of stationary solutions and periodic solutions to systems of parabolic PDEs in one space variable, using Chebyshev collocation in space. More precisely, the approximate solution is assumed of the form $u(x, t) = \sum_{k=0}^{n+1} u_k(t) \ell_k(x)$. Here $u_k(t)$ corresponds to $u(x_k, t)$ at the Chebyshev points $\{x_k\}_{k=1}^n$ with respect to the interval $[0, 1]$. The polynomials $\{\ell_k(x)\}_{k=0}^{n+1}$ are the Lagrange interpolating coefficients with respect to points $\{x_k\}_{k=0}^{n+1}$, where $x_0 = 0$ and $x_{n+1} = 1$. The number of Chebyshev points in $[0, 1]$, as well as the number of equations in the PDE system, can be set by the user in the file `brc.inc`.

As an illustrative application we consider the Brusselator (Holodniok, Knedlik & Kubíček (1987))

$$\begin{aligned} u_t &= D_x/L^2 u_{xx} + u^2 v - (B+1)u + A, \\ v_t &= D_y/L^2 v_{xx} - u^2 v + Bu, \end{aligned} \quad (16.3)$$

with boundary conditions $u(0, t) = u(1, t) = A$ and $v(0, t) = v(1, t) = B/A$.

Note that, given the non-adaptive spatial discretization, the computational procedure here is not appropriate for PDEs with solutions that rapidly vary in space, and care must be taken to recognize spurious solutions and bifurcations.

AUTO -COMMAND	ACTION
<code>! mkdir brc</code> <code>cd brc</code> <code>demo('brc')</code>	create an empty work directory change directory copy the demo files to the work directory
<code>run(c='brc.1')</code> <code>sv('brc')</code>	compute the stationary solution family with Hopf bifurcations save output-files as <code>b.brc</code> , <code>s.brc</code> , <code>d.brc</code>
<code>run(c='brc.2', s='brc')</code> <code>ap('brc')</code>	compute a family of periodic solutions from the first Hopf point. Constants changed : <code>IRS</code> , <code>IPS</code> append the output-files to <code>b.brc</code> , <code>s.brc</code> , <code>d.brc</code>
<code>run(c='brc.3', s='brc')</code> <code>ap('brc')</code>	compute a solution family from a sec- ondary periodic bifurcation. Constants changed : <code>IRS</code> , <code>ISW</code> append the output-files to <code>b.brc</code> , <code>s.brc</code> , <code>d.brc</code>

Table 16.4: Commands for running demo `brc`.

16.5 brf : Finite Differences in Space.

This demo illustrates the computation of stationary solutions and periodic solutions to systems of parabolic PDEs in one space variable. A fourth order accurate finite difference approximation is used to approximate the second order space derivatives. This reduces the PDE to an autonomous ODE of fixed dimension which AUTO is capable of treating. The spatial mesh is uniform; the number of mesh intervals, as well as the number of equations in the PDE system, can be set by the user in the file `brf.inc`.

As an illustrative application we consider the Brusselator (Holodniok, Knedlik & Kubíček (1987))

$$\begin{aligned} u_t &= D_x/L^2 u_{xx} + u^2 v - (B+1)u + A, \\ v_t &= D_y/L^2 v_{xx} - u^2 v + Bu, \end{aligned} \quad (16.4)$$

with boundary conditions $u(0, t) = u(1, t) = A$ and $v(0, t) = v(1, t) = B/A$.

Note that, given the non-adaptive spatial discretization, the computational procedure here is not appropriate for PDEs with solutions that rapidly vary in space, and care must be taken to recognize spurious solutions and bifurcations.

AUTO -COMMAND	ACTION
<code>! mkdir brf</code> <code>cd brf</code> <code>demo('brf')</code>	create an empty work directory change directory copy the demo files to the work directory
<code>run(c='brf.1')</code> <code>sv('brf')</code>	compute the stationary solution family with Hopf bifurcations save output-files as <code>b.brf</code> , <code>s.brf</code> , <code>d.brf</code>
<code>run(c='brf.2',s='brf')</code> <code>ap('brf')</code>	compute a family of periodic solutions from the first Hopf point. Constants changed : <code>IRS</code> , <code>IPS</code> append the output-files to <code>b.brf</code> , <code>s.brf</code> , <code>d.brf</code>
<code>run(c='brf.3',s='brf')</code> <code>ap('brf')</code>	compute a solution family from a sec- ondary periodic bifurcation. Constants changed : <code>IRS</code> , <code>ISW</code> append the output-files to <code>b.brf</code> , <code>s.brf</code> , <code>d.brf</code>

Table 16.5: Commands for running demo `brf`.

16.6 bru : Euler Time Integration (the Brusselator).

This demo illustrates the use of Euler's method for time integration of a nonlinear parabolic PDE. The example is the Brusselator (Holodniok, Knedlik & Kubíček (1987)), given by

$$\begin{aligned} u_t &= D_x/L^2 u_{xx} + u^2 v - (B+1)u + A, \\ v_t &= D_y/L^2 v_{xx} - u^2 v + Bu, \end{aligned} \tag{16.5}$$

with boundary conditions $u(0, t) = u(1, t) = A$ and $v(0, t) = v(1, t) = B/A$. All parameters are given fixed values for which a stable periodic solution is known to exist.

The continuation parameter is the independent time variable, namely `PAR(14)`. The `AUTO-`constants `DS`, `DSMIN`, and `DSMAX` then control the step size in space-time, here consisting of `PAR(14)` and $(u(x), v(x))$. Initial data at time zero are $u(x) = A - 0.5 \sin(\pi x)$ and $v(x) = B/A + 0.7 \sin(\pi x)$. Note that in the subroutine `STPNT` the space derivatives of u and v must also be provided; see the equations-file `bru.f`.

Euler time integration is only first order accurate, so that the time step must be sufficiently small to ensure correct results. This option has been added only as a convenience, and should generally be used only to locate stationary states. Indeed, in the case of the asymptotic periodic state of this demo, the number of required steps is very large and use of a better time integrator is advisable.

AUTO -COMMAND	ACTION
<code>! mkdir bru</code>	create an empty work directory
<code>cd bru</code>	change directory
<code>demo('bru')</code>	copy the demo files to the work directory
<code>run(c='bru.1')</code>	time integration
<code>sv('bru')</code>	save output-files as <code>b.bru</code> , <code>s.bru</code> , <code>d.bru</code>

Table 16.6: Commands for running demo `bru`.

Chapter 17

AUTO Demos : Optimization.

17.1 opt : A Model Algebraic Optimization Problem.

This demo illustrates the method of successive continuation for constrained optimization problems by applying it to the following simple problem : Find the maximum sum of coordinates on the unit sphere in R^5 . Coordinate 1 is treated as the state variable. Coordinates 2-5 are treated as control parameters. For details on the successive continuation procedure see Doedel, Keller & Kernévez (1991*a*), Doedel, Keller & Kernévez (1991*b*).

AUTO -COMMAND	ACTION
! mkdir opt cd opt demo('opt')	create an empty work directory change directory copy the demo files to the work directory
run(c='opt.1') sv('1')	one free equation parameter save output-files as b.1, s.1, d.1
run(c='opt.2',s='1') sv('2')	two free equation parameters; read restart data from s.1. Constants changed : IRS save output-files as b.2, s.2, d.2
run(c='opt.3',s='2') sv('3')	three free equation parameters; read restart data from s.2. Constants changed : IRS save output-files as b.3, s.3, d.3
run(c='opt.4',s='3') sv('4')	four free equation parameters; read restart data from s.3. Constants changed : IRS save output-files as b.4, s.4, d.4

Table 17.1: Commands for running demo opt.

17.2 ops : Optimization of Periodic Solutions.

This demo illustrates the method of successive continuation for the optimization of periodic solutions. For a detailed description of the basic method see Doedel, Keller & Kernévez (1991*b*). The illustrative system of autonomous ODEs, taken from Rodríguez-Luis (1991), is

$$\begin{aligned} x'(t) &= [-\lambda_4(x^3/3 - x) + (z - x)/\lambda_2 - y]/\lambda_1, \\ y'(t) &= x - \lambda_3, \\ z'(t) &= -(z - x)/\lambda_2, \end{aligned} \tag{17.1}$$

with objective functional

$$\omega = \int_0^1 g(x, y, z; \lambda_1, \lambda_2, \lambda_3, \lambda_4) dt,$$

where $g(x, y, z; \lambda_1, \lambda_2, \lambda_3, \lambda_4) \equiv \lambda_3$. Thus, in this application, a one-parameter extremum of g corresponds to a fold with respect to the problem parameter λ_3 , and multi-parameter extrema correspond to generalized folds. Note that, in general, the objective functional is an integral along the periodic orbit, so that a variety of optimization problems can be addressed.

For the case of periodic solutions, the extended optimality system can be generated automatically, i.e., one need only define the vector field and the objective functional, as in done in the file **ops.f**. For reference purpose it is convenient here to write down the full extended system in its general form :

$$\begin{aligned} u'(t) &= Tf(u(t), \lambda), \quad T \in \mathbb{R} \text{ (period)}, \quad u(\cdot), f(\cdot, \cdot) \in \mathbb{R}^n, \quad \lambda \in \mathbb{R}^{n_\lambda}, \\ w'(t) &= -Tf_u(u(t), \lambda)^* w(t) + \kappa u'_0(t) + \gamma g_u(u(t), \lambda)^*, \quad w(\cdot) \in \mathbb{R}^n, \quad \kappa, \gamma \in \mathbb{R}, \\ u(1) - u(0) &= 0, \quad w(1) - w(0) = 0, \\ \int_0^1 u(t)^* u'_0(t) dt &= 0, \\ \int_0^1 \omega - g(u(t), \lambda) dt &= 0, \\ \int_0^1 w(t)^* w(t) + \kappa^2 + \gamma^2 - \alpha dt &= 0, \quad \alpha \in \mathbb{R}, \\ \int_0^1 f(u(t), \lambda)^* w(t) - \gamma g_T(u(t), \lambda) - \tau_0 dt &= 0, \quad \tau_0 \in \mathbb{R}, \\ \int_0^1 Tf_{\lambda_i}(u(t), \lambda)^* w(t) - \gamma g_{\lambda_i}(u(t), \lambda) - \tau_i dt &= 0, \quad \tau_i \in \mathbb{R}, \quad i = 1, \dots, n_\lambda. \end{aligned} \tag{17.2}$$

Above u_0 is a reference solution, namely, the previous solution along a solution family.

In the computations below, the two preliminary runs, with $\text{IPS}=1$ and $\text{IPS}=2$, respectively, locate periodic solutions. The subsequent runs are with $\text{IPS}=15$ and hence use the automatically generated extended system.

- **Run 1.** Locate a Hopf bifurcation. The free system parameter is λ_3 .
- **Run 2.** Compute a family of periodic solutions from the Hopf bifurcation.
- **Run 3.** This run retraces part of the periodic solution family, using the full optimality system, but with all adjoint variables, $w(\cdot), \kappa, \gamma$, and hence α , equal to zero. The optimality parameters τ_0 and τ_3 are zero throughout. An extremum of the objective functional with respect to λ_3 is located. Such a point corresponds to a branch point of the extended system. Given the choice of objective functional in this demo, this extremum is also a fold with respect to λ_3 .
- **Run 4.** Branch switching at the above-found branch point yields nonzero values of the adjoint variables. Any point on the bifurcating family away from the branch point can serve as starting solution for the next run. In fact, the branch-switching can be viewed as generating a nonzero eigenvector in an eigenvalue-eigenvector relation. Apart from the adjoint variables, all other variables remain unchanged along the bifurcating family.
- **Run 5.** The above-found starting solution is continued in two system parameters, here λ_3 and λ_2 ; i.e., a two-parameter family of extrema with respect to λ_3 is computed. Along this family the value of the optimality parameter τ_2 is monitored, i.e., the value of the functional that vanishes at an extremum with respect to the system parameter λ_2 . Such a zero of τ_2 is, in fact, located, and hence an extremum of the objective functional with respect to both λ_2 and λ_3 has been found. Note that, in general, τ_i is the value of the functional that vanishes at an extremum with respect to the system parameter λ_i .
- **Run 6.** In the final run, the above-found two-parameter extremum is continued in three system parameters, here λ_1, λ_2 , and λ_3 , toward $\lambda_1 = 0$. Again, given the particular choice of objective functional, this final continuation has an alternate significance here : it also represents a three-parameter family of transcritical secondary periodic bifurcations points.

Although not illustrated here, one can restart an ordinary continuation of periodic solutions, using $\text{IPS}=2$ or $\text{IPS}=3$, from a labeled solution point on a family computed with $\text{IPS}=15$.

The free scalar variables specified in the AUTO constants-files for Run 3 and Run 4 are shown in Table 17.2.

Index	3	11	12	22	-22	-23	-31
Variable	λ_3	T	α	τ_2	$[\lambda_2]$	$[\lambda_3]$	$[T]$

Table 17.2: Runs 3 and 4 (files `c.ops.3` and `c.ops.4`).

The parameter α , which is the norm of the adjoint variables, becomes nonzero after branch switching in Run 4. The negative indices (-22, -23, and -31) set the active optimality functionals, namely for λ_2 , λ_3 , and T , respectively, with corresponding variables τ_2 , τ_3 , and τ_0 , respectively. These should be set in the first run with `IPS=15` and remain unchanged in all subsequent runs.

Index	3	2	11	22	-22	-23	-31
Variable	λ_3	λ_2	T	τ_2	$[\lambda_2]$	$[\lambda_3]$	$[T]$

Table 17.3: Run 5 (file `c.ops.5`).

In Run 5 the parameter α , which has been replaced by λ_2 , remains fixed and nonzero. The variable τ_2 monitors the value of the optimality functional associated with λ_2 . The zero of τ_2 located in this run signals an extremum with respect to λ_2 .

Index	3	2	1	11	-22	-23	-31
Variable	λ_3	λ_2	λ_1	T	$[\lambda_2]$	$[\lambda_3]$	$[T]$

Table 17.4: Run 6 (file `c.ops.6`).

In Run 6 τ_2 , which has been replaced by λ_1 , remains zero.

Note that τ_0 and τ_3 are not used as variables in any of the runs; in fact, their values remain zero throughout. Also note that the optimality functionals corresponding to τ_0 and τ_3 (or, equivalently, to T and λ_3) are active in all runs. This set-up allows the detection of the extremum of the objective functional, with T and λ_3 as scalar equation parameters, as a bifurcation in the third run.

The parameter λ_4 , and its corresponding optimality variable τ_4 , are not used in this demo. Also, λ_1 is used in the last run only, and its corresponding optimality variable τ_1 is never used.

AUTO -COMMAND	ACTION
! mkdir ops cd ops demo('ops')	create an empty work directory change directory copy the demo files to the work directory
run(c='ops.1') sv('0')	locate a Hopf bifurcation save output-files as b.0, s.0, d.0
run(c='ops.2',s='0') ap('0')	compute a family of periodic solutions; restart from s.0. Constants changed : IPS, IRS, NMX, NUZR append the output-files to b.0, s.0, d.0
run(c='ops.3',s='0') sv('1')	locate a 1-parameter extremum as a bi- furcation; restart from s.0. Constants changed : IPS, IRS, ICP, ... save the output-files as b.1, s.1, d.1
run(c='ops.4',s='1') ap('1')	switch branches to generate optimality starting data; restart from s.1. Constants changed : IRS, ISP, ISW, NMX append the output-files to b.1, s.1, d.1
run(c='ops.5',s='1') sv('2')	compute 2-parameter family of 1- parameter extrema; restart from s.1. Constants changed : IRS, ISW, ICP, ISW, ... save the output-files as b.2, s.2, d.2
run(c='ops.6',s='2') sv('3')	compute 3-parameter family of 2- parameter extrema; restart from s.2. Constants changed : IRS, ICP, EPSL, EPSU, NUZR save the output-files as b.3, s.3, d.3

Table 17.5: Commands for running demo ops.

17.3 obv : Optimization for a BVP.

This demo illustrates use of the method of successive continuation for a boundary value optimization problem. A detailed description of the basic method, as well as a discussion of the specific application considered here, is given in Doedel, Keller & Kernévez (1991*b*). The required extended system is fully programmed here in the user-supplied routines in `obv.f`. For the case of periodic solutions the optimality system can be generated automatically; see the demo `ops`.

Consider the system

$$\begin{aligned} u_1'(t) &= u_2(t), \\ u_2'(t) &= -\lambda_1 e^{p(u_1, \lambda_2, \lambda_3)}, \end{aligned} \quad (17.3)$$

where $p(u_1, \lambda_2, \lambda_3) \equiv u_1 + \lambda_2 u_1^2 + \lambda_3 u_1^4$, with boundary conditions

$$\begin{aligned} u_1(0) &= 0, \\ u_1(1) &= 0. \end{aligned} \quad (17.4)$$

The objective functional is

$$\omega = \int_0^1 (u_1(t) - 1)^2 dt + \frac{1}{10} \sum_{k=1}^3 \lambda_k^2.$$

The successive continuation equations are given by

$$\begin{aligned} u_1'(t) &= u_2(t), \\ u_2'(t) &= -\lambda_1 e^{p(u_1, \lambda_2, \lambda_3)}, \\ w_1'(t) &= \lambda_1 e^{p(u_1, \lambda_2, \lambda_3)} p_{u_1} w_2(t) + 2\gamma(u_1(t) - 1), \\ w_2'(t) &= -w_1(t), \end{aligned} \quad (17.5)$$

where

$$p_{u_1} \equiv \frac{\partial p}{\partial u_1} = 1 + 2\lambda_2 u_1 + 4\lambda_3 u_1^3,$$

with

$$\begin{aligned} u_1(0) &= 0, & w_1(0) - \beta_1 &= 0, & w_2(0) &= 0, \\ u_1(1) &= 0, & w_1(1) + \beta_2 &= 0, & w_2(1) &= 0, \end{aligned} \quad (17.6)$$

$$\int_0^1 \left[\omega - (u_1(t) - 1)^2 - \frac{1}{10} \sum_{k=1}^3 \lambda_k^2 \right] dt = 0,$$

$$\int_0^1 [w_1^2(t) - \alpha_0] dt = 0,$$

$$\begin{aligned} \int_0^1 [-e^{p(u_1, \lambda_2, \lambda_3)} w_2(t) - \frac{1}{5} \gamma \lambda_1] dt &= 0, \\ \int_0^1 [-\lambda_1 e^{p(u_1, \lambda_2, \lambda_3)} u_1(t)^2 w_2(t) - \frac{1}{5} \gamma \lambda_2 - \tau_2] dt &= 0, \\ \int_0^1 [-\lambda_1 e^{p(u_1, \lambda_2, \lambda_3)} u_1(t)^4 w_2(t) - \frac{1}{5} \gamma \lambda_3 - \tau_3] dt &= 0. \end{aligned} \quad (17.7)$$

In the first run the free equation parameter is λ_1 . All adjoint variables are zero. Three extrema of the objective function are located. These correspond to branch points and, in the second run, branch switching is done at one of these. Along the bifurcating family the adjoint

variables become nonzero, while state variables and λ_1 remain constant. Any such non-trivial solution point can be used for continuation in two equation parameters, after fixing the L_2 -norm of one of the adjoint variables. This is done in the third run. Along the resulting family several two-parameter extrema are located by monitoring certain inner products. One of these is further continued in three equation parameters in the final run, where a three-parameter extremum is located.

AUTO -COMMAND	ACTION
! mkdir obv cd obv demo('obv')	create an empty work directory change directory copy the demo files to the work directory
run(c='obv.1') sv('obv')	locate 1-parameter extrema as branch points save output-files as b.obv, s.obv, d.obv
run(c='obv.2',s='obv') sv('1')	compute a few step on the first bifurcating family. Constants changed : IRS, ISW, NMX save the output-files as b.1, s.1, d.1
run(c='obv.3',s='1') sv('2')	locate 2-parameter extremum; restart from s.1. Constants changed : IRS, ISW, NMX, ICP(3) save the output-files as b.2, s.2, d.2
run(c='obv.4',s='2') sv('3')	locate 3-parameter extremum; restart from s.2. Constants changed : IRS, ICP(4) save the output-files as b.3, s.3, d.3

Table 17.6: Commands for running demo obv.

Chapter 18

AUTO Demos : Connecting orbits.

18.1 fsh : A Saddle-Node Connection.

This demo illustrates the computation of travelling wave front solutions to the Fisher equation,

$$\begin{aligned} w_t &= w_{xx} + f(w), & -\infty < x < \infty, \quad t > 0, \\ f(w) &\equiv w(1 - w). \end{aligned} \tag{18.1}$$

We look for solutions of the form $w(x, t) = u(x + ct)$, where c is the wave speed. This gives the first order system

$$\begin{aligned} u_1'(z) &= u_2(z), \\ u_2'(z) &= cu_2(z) - f(u_1(z)). \end{aligned} \tag{18.2}$$

Its fixed point $(0, 0)$ has two positive eigenvalues when $c > 2$. The other fixed point, $(1, 0)$, is a saddle point. A family of orbits connecting the two fixed points requires one free parameter; see Friedman & Doedel (1991). Here we take this parameter to be the wave speed c .

In the first run a starting connecting orbit is computed by continuation in the period T . This procedure can be used generally for time integration of an ODE with AUTO . Starting data in **STPNT** correspond to a point on the approximate stable manifold of $(1, 0)$, with T small. In this demo the “free” end point of the orbit necessary approaches the unstable fixed point $(0, 0)$. A computed orbit with sufficiently large T is then chosen as restart orbit in the second run, where, typically, one replaces T by c as continuation parameter. However, in the second run below, we also add a phase condition, and both c and T remain free.

AUTO -COMMAND	ACTION
<code>! mkdir fsh</code>	create an empty work directory
<code>cd fsh</code>	change directory
<code>demo('fsh')</code>	copy the demo files to the work directory
<code>run(c='fsh.1')</code> <code>sv('0')</code>	continuation in the period T , with c fixed; no phase condition save output-files as <code>b.0</code> , <code>s.0</code> , <code>d.0</code>
<code>run(c='fsh.2',s='0')</code> <code>sv('fsh')</code>	continuation in c and T , with active phase condition. Constants changed : <code>IRS</code> , <code>ICP</code> , <code>NINT</code> , <code>DS</code> save output-files as <code>b.fsh</code> , <code>s.fsh</code> , <code>d.fsh</code>

Table 18.1: Commands for running demo `fsh`.

18.2 nag : A Saddle-Saddle Connection.

This demo illustrates the computation of traveling wave front solutions to Nagumo's equation,

$$\begin{aligned} w_t &= w_{xx} + f(w, a), & -\infty < x < \infty, \quad t > 0, \\ f(w, a) &\equiv w(1-w)(w-a), & 0 < a < 1. \end{aligned} \quad (18.3)$$

We look for solutions of the form $w(x, t) = u(x + ct)$, where c is the wave speed. This gives the first order system

$$\begin{aligned} u_1'(z) &= u_2(z), \\ u_2'(z) &= cu_2(z) - f(u_1(z), a), \end{aligned} \quad (18.4)$$

where $z = x + ct$, and $' = d/dz$. If $a = 1/2$ and $c = 0$ then there are two analytically known heteroclinic connections, one of which is given by

$$u_1(z) = \frac{e^{\frac{1}{2}\sqrt{2}z}}{1 + e^{\frac{1}{2}\sqrt{2}z}}, \quad u_2(z) = u_1'(z), \quad -\infty < z < \infty.$$

The second heteroclinic connection is obtained by reflecting the phase plane representation of the first with respect to the u_1 -axis. In fact, the two connections together constitute a heteroclinic cycle. One of the exact solutions is used below as starting orbit. To start from the second exact solution, change SIGN=-1 in the routine **STPNT** in **nag.f** and repeat the computations below; see also Friedman & Doedel (1991).

AUTO -COMMAND	ACTION
! mkdir nag	create an empty work directory
cd nag	change directory
demo('nag')	copy the demo files to the work directory
run(c='nag.1') sv('nag')	compute part of first family of heteroclinic orbits save output-files as b.nag, s.nag, d.nag
run(c='nag.2', s='nag')	compute first family in opposite direction. Constants changed : DS
ap('nag')	append output-files to b.nag, s.nag, d.nag

Table 18.2: Commands for running demo **nag**.

18.3 stw : Continuation of Sharp Traveling Waves.

This demo illustrates the computation of sharp traveling wave front solutions to nonlinear diffusion problems of the form

$$w_t = A(w)w_{xx} + B(w)w_x^2 + C(w),$$

with $A(w) = a_1w + a_2w^2$, $B(w) = b_0 + b_1w + b_2w^2$, and $C(w) = c_0 + c_1w + c_2w^2$. Such equations can have **sharp traveling wave fronts** as solutions, i.e., solutions of the form $w(x, t) = u(x + ct)$ for which there is a z_0 such that $u(z) = 0$ for $z \geq z_0$, $u(z) \neq 0$ for $z < z_0$, and $u(z) \rightarrow \text{constant}$ as $z \rightarrow -\infty$. These solutions are actually generalized solutions, since they need not be differentiable at z_0 .

Specifically, in this demo a homotopy path will be computed from an analytically known exact sharp traveling wave solution of

$$(1) \quad w_t = 2ww_{xx} + 2w_x^2 + w(1 - w),$$

to a corresponding sharp traveling wave of

$$(2) \quad w_t = (2w + w^2)w_{xx} + ww_x^2 + w(1 - w).$$

This problem is also considered in Doedel, Keller & Kernévez (1991*b*). For these two special cases the functions A, B, C are defined by the coefficients in Table 18.3.

	a_1	a_2	b_0	b_1	b_2	c_0	c_1	c_2
Case (1)	2	0	2	0	0	0	1	-1
Case (2)	2	1	0	1	0	0	1	-1

Table 18.3: Problem coefficients in demo **stw**.

With $w(x, t) = u(x + ct)$, $z = x + ct$, one obtains the reduced system

$$\begin{aligned} u_1'(z) &= u_2, \\ u_2'(z) &= [cu_2 - B(u_1)u_2^2 - C(u_1)]/A(u_1). \end{aligned} \tag{18.5}$$

To remove the singularity when $u_1 = 0$, we apply a nonlinear transformation of the independent variable (see Aronson (1980)), viz., $d/d\tilde{z} = A(u_1)d/dz$, which changes the above equation into

$$\begin{aligned} u_1'(\tilde{z}) &= A(u_1)u_2, \\ u_2'(\tilde{z}) &= cu_2 - B(u_1)u_2^2 - C(u_1). \end{aligned} \tag{18.6}$$

Sharp traveling waves then correspond to heteroclinic connections in this transformed system.

Finally, we map $[0, T] \rightarrow [0, 1]$ by the transformation $\xi = \tilde{z}/T$. With this scaling of the independent variable, the reduced system becomes

$$\begin{aligned} u_1'(\xi) &= TA(u_1)u_2, \\ u_2'(\xi) &= T[cu_2 - B(u_1)u_2^2 - C(u_1)]. \end{aligned} \quad (18.7)$$

For Case 1 this equation has a known exact solution, namely,

$$u(\xi) = \frac{1}{1 + \exp(T\xi)}, \quad v(\xi) = \frac{-\frac{1}{2}}{1 + \exp(-T\xi)}.$$

This solution has wave speed $c = 1$. In the limit as $T \rightarrow \infty$ its phase plane trajectory connects the stationary points $(1, 0)$ and $(0, -\frac{1}{2})$.

The sharp traveling wave in Case 2 can now be obtained using the following homotopy. Let $(a_1, a_2, b_0, b_1, b_2) = (1 - \lambda)(2, 0, 2, 0, 0) + \lambda(2, 1, 0, 1, 0)$. Then as λ varies continuously from 0 to 1, the parameters $(a_1, a_2, b_0, b_1, b_2)$ vary continuously from the values for Case 1 to the values for Case 2.

AUTO -COMMAND	ACTION
<code>! mkdir stw</code>	create an empty work directory
<code>cd stw</code>	change directory
<code>demo('stw')</code>	copy the demo files to the work directory
<code>run(c='stw.1')</code>	continuation of the sharp traveling wave
<code>sv('stw')</code>	save output-files as <code>b.stw</code> , <code>s.stw</code> , <code>d.stw</code>

Table 18.4: Commands for running demo `stw`.

Chapter 19

AUTO Demos : Miscellaneous.

19.1 pvl : Use of the Routine PVLS.

Consider Bratu's equation

$$\begin{aligned} u_1' &= u_2, \\ u_2' &= -p_1 e^{u_1}, \end{aligned} \tag{19.1}$$

with boundary conditions $u_1(0) = 0$, $u_1(1) = 0$. As in demo `exp`, a solution curve requires one free parameter; here p_1 .

Note that additional parameters are specified in the user-supplied subroutine **PVLS** in file `pvl.s.f`, namely, p_2 (the L_2 -norm of u_1), p_3 (the minimum of u_2 on the space-interval $[0, 1]$), p_4 (the boundary value $u_2(0)$). These additional parameters should be considered as “solution measures” for output purposes; they should not be treated as true continuation parameters.

Note also that four free parameters are specified in the AUTO -constants file `c.pvl.1`, namely, p_1 , p_2 , p_3 , and p_4 . The first one in this list, p_1 , is the true continuation parameter. The parameters p_2 , p_3 , and p_4 are *overspecified* so that their values will appear in the output. However, *it is essential that the true continuation parameter appear first*. For example, it would be an error to specify the parameters in the following order : p_2, p_1, p_3, p_4 .

In general, true continuation parameters must appear first in the parameter-specification in the AUTO constants-file. Overspecified parameters will be printed, and can be defined in **PVLS**, but they are not part of the intrinsic continuation procedure.

As this demo also illustrates (see the `UZR` values in `c.pvl.1`), labeled solutions can also be output at selected values of the overspecified parameters.

AUTO -COMMAND	ACTION
<code>! mkdir pvl</code>	create an empty work directory
<code>cd pvl</code>	change directory
<code>demo('pvl')</code>	copy the demo files to the work directory
<code>run(c='pvl.1')</code>	compute a solution family
<code>sv('pvl')</code>	save output-files as <code>b.pvl</code> , <code>s.pvl</code> , <code>d.pvl</code>

Table 19.1: Commands for running demo `pvl`.

19.2 ext : Spurious Solutions to BVB.

This demo illustrates the computation of spurious solutions to the boundary value problem

$$\begin{aligned} u_1' - u_2 &= 0, \\ u_2' + \lambda^2 \pi^2 \sin(u_1 + u_1^2 + u_1^3) &= 0, \quad t \in [0, 1], \\ u_1(0) &= 0, \quad u_1(1) = 0. \end{aligned} \tag{19.2}$$

Here the differential equation is discretized using a fixed uniform mesh. This results in spurious solutions that disappear when an adaptive mesh is used. See the AUTO -constant `IAD` in Section 10.3. This example is also considered in Beyn & Doedel (1981) and Doedel, Keller & Kernévez (1991*b*).

AUTO -COMMAND	ACTION
<code>! mkdir ext</code>	create an empty work directory
<code>cd ext</code>	change directory
<code>demo('ext')</code>	copy the demo files to the work directory
<code>run(c='ext.1')</code> <code>sv('ext')</code>	detect bifurcations from the trivial solution family save output-files as <code>b.ext</code> , <code>s.ext</code> , <code>d.ext</code>
<code>run(c='ext.2',s='ext')</code>	compute a bifurcating family containing spurious bifurcations. Constants changed : <code>IRS</code> , <code>ISW</code> , <code>NUZR</code>
<code>ap('ext')</code>	append output-files to <code>b.ext</code> , <code>s.ext</code> , <code>d.ext</code>

Table 19.2: Commands for running demo `ext`.

19.3 tim : A Test Problem for Timing AUTO .

This demo is a boundary value problem with variable dimension `NDIM`. It can be used to time the performance of AUTO for various choices of `NDIM` (which must be even), `NTST`, and `NCOL`. The equations are

$$\begin{aligned} u'_i &= u_i, \\ v'_i &= -p_1 e(u_i), \end{aligned} \tag{19.3}$$

$i = 1, \dots, \text{NDIM}/2$, with boundary conditions $u_i(0) = 0$, $u_i(1) = 0$. Here

$$e(u) = \sum_{k=0}^n \frac{u^k}{k!},$$

with $n = 25$. The computation requires 10 full *LU*-decompositions of the linearized system that arises from Newton's method for solving the collocation equations. The commands for running the timing problem for a particular choice of `NDIM`, `NTST`, and `NCOL` are given below. (Note that if `NDIM` is changed then `NBC` must be changed accordingly.)

AUTO -COMMAND	ACTION
<code>! mkdir tim</code>	create an empty work directory
<code>cd tim</code>	change directory
<code>demo('tim')</code>	copy the demo files to the work directory
<code>run(c='tim.1')</code>	Timing run
<code>sv('tim')</code>	save output-files as <code>b.tim</code> , <code>s.tim</code> , <code>d.tim</code>

Table 19.3: Commands for running demo `tim`.

Chapter 20

HomCont.

20.1 Introduction.

HomCont is a collection of routines for the continuation of homoclinic solutions to ODEs in two or more parameters. The accurate detection and multi-parameter continuation of certain codimension-two singularities is allowed for, including all known cases that involve a unique homoclinic orbit at the singular point. Homoclinic connections to hyperbolic and non-hyperbolic equilibria are allowed as are certain heteroclinic orbits. Homoclinic orbits in reversible systems can also be computed. The theory behind the methods used is explained in Champneys & Kuznetsov (1994), Bai & Champneys (1996), Sandstede (1995*b*, 1995*c*), Champneys, Kuznetsov & Sandstede (1996) and references therein. The final cited paper contains a concise description of the present version.

The current implementation of HomCont must be considered as experimental, and updates are anticipated. The HomCont routines are in the file `auto/07p/src/autlib5.f`. Expert users wishing to modify the routines may look there. Note also that at present, HomCont can be run only in AUTO Command Mode and not with the GUI.

20.2 HomCont Files and Routines.

In order to run HomCont one must prepare an equations file `xxx.f`, where `xxx` is the name of the example, and two constants-files `c.xxx` and `h.xxx`. The first two of these files are in the standard AUTO format, whereas the `h.xxx` file contains constants that are specific to homoclinic continuation. The choice `IPS=9` in `c.xxx` specifies the problem as being homoclinic continuation, in which case `h.xxx` is required.

The equation-file `kpr.f` serves as a sample for new equation files. It contains the Fortran routines **FUNC**, **STPNT**, **PVLS**, **BCND**, **ICND** and **FOPT**. The final three are dummy routines which are never needed for homoclinic continuation. Note a minor difference in **STPNT** and **PVLS** with other AUTO equation-files, in that the common block `/BLHOM/` is required.

The constants-file `c.xxx` is identical in format to other AUTO constants-files. Note that the values of the constants `NBC` and `NINT` are irrelevant, as these are set automatically by the choice `IPS=9`. Also, the choice `JAC=1` is strongly recommended, because the Jacobian is used

extensively for calculating the linearization at the equilibria and hence for evaluating boundary conditions and certain test functions. However, note that `JAC=1` does not necessarily mean that auto will use the analytically specified Jacobian for continuation.

20.3 HomCont-Constants.

An example for the additional file `h.xxx` is listed below:

```

1 2 1 1 1    NUNSTAB,NSTAB,IEQUIB,ITWIST,ISTART
0            NREV,(/,I,IREV(I)),I=1,NREV)
1            NFIXED,(/,I,IFIXED(I)),I=1,NFIXED)
13
1            NPSI,(/,I,IPSI(I)),I=1,NPSI)
9 10 13
```

The constants specified in `h.xxx` have the following meaning.

20.3.1 NUNSTAB

Number of unstable eigenvalues of the left-hand equilibrium (the equilibrium approached by the orbit as $t \rightarrow -\infty$).

20.3.2 NSTAB

Number of stable eigenvalues of the right-hand equilibrium (the equilibrium approached by the orbit as $t \rightarrow +\infty$).

20.3.3 IEQUIB

- `IEQUIB=0` : Homoclinic orbits to hyperbolic equilibria; the equilibrium is specified explicitly in `PVLS` and stored in `PAR(11+I)`, `I=1,NDIM`.
- `IEQUIB=1` : Homoclinic orbits to hyperbolic equilibria; the equilibrium is solved for during continuation. Initial values for the equilibrium are stored in `PAR(11+I)`, `I=1,NDIM` in `STPNT`.
- `IEQUIB=2` : Homoclinic orbits to a saddle-node; initial values for the equilibrium are stored in `PAR(11+I)`, `I=1,NDIM` in `STPNT`.
- `IEQUIB=-1` : Heteroclinic orbits to hyperbolic equilibria; the equilibria are specified explicitly in `PVLS` and stored in `PAR(11+I)`, `I=1,NDIM` (left-hand equilibrium) and `PAR(11+I)`, `I=NDIM+1,2*NDIM` (right-hand equilibrium).
- `IEQUIB=-2` : Heteroclinic orbits to hyperbolic equilibria; the equilibria are solved for during continuation. Initial values are specified in `STPNT` and stored in `PAR(11+I)`, `I=1,NDIM` (left-hand equilibrium), `PAR(11+I)`, `I=NDIM+1,2*NDIM` (right-hand equilibrium).

20.3.4 ITWIST

- ITWIST=0 : the orientation of the homoclinic orbit is not computed.
- ITWIST=1 : the orientation of the homoclinic orbit is computed. For this purpose, the adjoint variational equation is solved for the unique bounded solution. If IRS = 0, an initial solution to the adjoint equation must be specified as well. However, if IRS>0 and ITWIST has just been increased from zero, then AUTO will automatically generate the initial solution to the adjoint. In this case, a dummy Newton-step should be performed, see Section 20.7 for more details.

20.3.5 ISTART

- ISTART=1 : This option is obsolete in the current version. It may be used as a flag that a solution is to be restarted from a previously computed point or from numerical data converted into AUTO format using `us` or `@fc`. In this case IRS>0.
- ISTART=2 : If IRS=0, an explicit solution must be specified in the subroutine `STPNT` in the usual format.
- ISTART=3 : The “homotopy” approach is used for starting, see Section 20.7 for more details. Note that this is not available with the choice IEQUIB=2.
- ISTART=4 : A phase-shift is performed for homoclinic orbits to let the equilibrium (either fixed or non-fixed, depending on IEQUIB) correspond to $t = 0$ and $t = 1$. This is necessary if a periodic orbit that is close to a homoclinic orbit is continued into a homoclinic orbit.
- ISTART=-N, $N = 1, 2, 3, \dots$: Homoclinic branch switching: this description is for reference only. We refer to the demo in Chapter 27 to see how this can be used in actual practice and to Oldeman, Champneys & B. (2003) for theory and background.

The orbit is split into $N + 1$ parts and AUTO sees it as an $(N + 1) \times \text{NDIM}$ -dimensional object. The first part u_0 goes from the equilibrium to the point x_0 that is furthest from the equilibrium. Then follow $N - 1$ shifted copies of the orbit, which travel from the point x_0 back to the point x_0 . The last part U_N goes from the point x_0 back to the equilibrium. The derivatives \dot{x}_0 with respect to time of the point that is furthest from the equilibrium are stored at the parameters `PAR(NPARX-NDIM+1...NPARX)`.

If ITWIST=1, and this was also the case in the preceding run, then a copy of the adjoint vector Ψ at x_0 is stored at the parameters `PAR(NPARX-NDIM*2+1...NPARX-NDIM)` and Lin’s method can be used to do homoclinic branch switching. To be more precise, the individual parts u_i and u_{i+1} are at distances ε_i away from each other, along the Lin vector Psi , at the left- and right-hand end points. These gaps ε_i are at parameters `PAR(20+2*i)`. Moreover, each part (except u_{N+1}) ends at at a Poincaré section which goes through x_0 and is perpendicular to \dot{x}_0 .

The times T_i that each part u_i takes are stored as follows: $T_0 = \text{PAR}(10)$, $T_N = \text{PAR}(11)$ and $T_i = \text{PAR}(19+2*i)$ for $i = 1 \dots N-1$. Through a continuation in problem parameters, gaps ε_i , and times T_i it is possible to switch from a 1-homoclinic to an N -homoclinic orbit.

If `ITWIST=0`, the adjoint vector is not computed and Lin's method is not used. Instead, AUTO produces a gap $\varepsilon = \text{PAR}(22)$ at the right-hand end point p of u_{N+1} , measuring the distance between the stable manifold of the equilibrium and p . This technique can also be used to find 2-homoclinic orbits, by varying in ε and T_1 , similar to the method described before, but only if the unstable manifold is one-dimensional. Because this method is more limited than the method using Lin vectors, we do not recommend it for normal usage.

To switch back to a normal homoclinic orbit, set `ISTART` back to a positive value such as 1. Now HomCont has lost all the information about the adjoint, so if `ITWIST` is set to 0, HomCont does a normal continuation without the adjoint, and if `ITWIST` is set to 1, one needs to do a Newton dummy step first to recalculate the adjoint.

20.3.6 NREV, IREV

If `NREV=1` then it is assumed that the system is reversible under the transformation $t \rightarrow -t$ and $U(i) \rightarrow -U(i)$ for all i with `IREV(i)>0`. Then only half the homoclinic solution is solved for with right-hand boundary conditions specifying that the solution is symmetric under the reversibility (see Champneys & Spence (1993)). The number of free parameters is then reduced by one. Otherwise `IREV=0`.

20.3.7 NFIXED, IFIXED

Number and labels of test functions that are held fixed. E.g., with `NFIXED=1` one can compute a locus in one extra parameter of a singularity defined by test function `PSI(IFIXED(1))=0`.

20.3.8 NPSI, IPSI

Number and labels of activated test functions for detecting homoclinic bifurcations, see Section 20.6 for a list. If a test function is activated then the corresponding parameter (`IPSI(I)+20`) must be added to the list of continuation parameters `NICP, (ICP(I), I=1 NICP)` and zero of this parameter added to the list of user-defined output points `NUZR, (/ , I, PAR(I)), I=1, NUZR` in `c.xxx`.

20.4 Restrictions on HomCont Constants.

Note that certain combinations of these constants are not allowed in the present implementation. In particular,

- The computation of orientation `ITWIST=1` is not implemented for `IEQUIB<0` (heteroclinic orbits), `IEQUIB=2` (saddle-node homoclinics), `IREV=1` (reversible systems), `ISTART=3` (homotopy method for starting), or if the equilibrium contains complex eigenvalues in its linearization.
- The homotopy method `ISTART=3` is not fully implemented for heteroclinic connections `IEQUIB<0`, saddle-node homoclinic orbits `IEQUIB=2` or reversible systems `IREV=1`.

- Certain test functions are not valid for certain forms of continuation (see Section 20.6 below); for example `PSI(13)` and `PSI(14)` only make sense if `ITWIST=1` and `PSI(15)` and `PSI(16)` only apply to `IEQUIB=2`.

20.5 Restrictions on the Use of PAR.

The parameters `PAR(1)` – `PAR(9)` can be used freely by the user. The other parameters are used as follows :

- `PAR(11)` : The value of `PAR(11)` equals the length of the time interval over which a homoclinic solution is computed. Also referred to as “period”. This must be specified in `STPNT`.
- `PAR(10)` : If `ITWIST=1` then `PAR(10)` is used internally as a dummy parameter so that the adjoint equation is well-posed.
- `PAR(12)`–`PAR(20)` : These are used for specifying the equilibria and (if `ISTART=3`) the artificial parameters of the homotopy method (see Section 20.7 below).
- `PAR(21)`–`PAR(36)` : These parameters are used for storing the test functions (see Section 20.6).

The output is in an identical format to AUTO except that additional information at each computed point is written in `fort.9`. This information comprises the eigenvalues of the (left-hand) equilibrium, the values of each activated test function and, if `ITWIST=1`, whether the saddle homoclinic loop is orientable or not. Note that the statement about orientability is only meaningful if the leading eigenvalues are not complex and the homoclinic solution is not in a flip configuration, that is, none of the test functions ψ_i for $i = 11, 12, 13, 14$ is zero (or close to zero), see Section 20.6. Finally, the values of the `NPSI` activated test functions are written.

20.6 Test Functions.

Codimension-two homoclinic orbits are detected along branches of codim 1 homoclinics by locating zeroes of certain test functions ψ_i . The test functions that are “switched on” during any continuation are given by the choice of the labels i , and are specified by the parameters `NPSI, (/ , I, IPSI(I)), I=1, NPSI` in `h.xxx`. Here `NPSI` gives the number of activated test functions and `IPSI(1), ..., IPSI(NPSI)` give the labels of the test functions (numbers between 1 and 16). A zero of each labeled test function defines a certain codimension-two homoclinic singularity, specified as follows. The notation used for eigenvalues is the same as that in Champneys & Kuznetsov (1994) or Champneys et al. (1996).

- $i = 1$: Resonant eigenvalues (neutral saddle); $\mu_1 = -\lambda_1$.
- $i = 2$: Double real leading stable eigenvalues (saddle to saddle-focus transition); $\mu_1 = \mu_2$.

- $i = 3$: Double real leading unstable eigenvalues (saddle to saddle-focus transition); $\lambda_1 = \lambda_2$.
- $i = 4$: Neutral saddle, saddle-focus or bi-focus (includes $i = 1$); $\text{Re}(\mu_1) = -\text{Re}(\lambda_1)$.
- $i = 5$: Neutrally-divergent saddle-focus (stable eigenvalues complex); $\text{Re}(\lambda_1) = -\text{Re}(\mu_1) - \text{Re}(\mu_2)$.
- $i = 6$: Neutrally-divergent saddle-focus (unstable eigenvalues complex); $\text{Re}(\mu_1) = -\text{Re}(\lambda_1) - \text{Re}(\lambda_2)$.
- $i = 7$: Three leading eigenvalues (stable); $\text{Re}(\lambda_1) = -\text{Re}(\mu_1) - \text{Re}(\mu_2)$.
- $i = 8$: Three leading eigenvalues (unstable); $\text{Re}(\mu_1) = -\text{Re}(\lambda_1) - \text{Re}(\lambda_2)$.
- $i = 9$: Local bifurcation (zero eigenvalue or Hopf): number of stable eigenvalues decreases; $\text{Re}(\mu_1) = 0$.
- $i = 10$: Local bifurcation (zero eigenvalue or Hopf): number of unstable eigenvalues decreases; $\text{Re}(\lambda_1) = 0$.
- $i = 11$: Orbit flip with respect to leading stable direction (e.g., 1D unstable manifold).
- $i = 12$: Orbit flip with respect to leading unstable direction, (e.g., 1D stable manifold).
- $i = 13$: Inclination flip with respect to stable manifold (e.g., 1D unstable manifold).
- $i = 14$: Inclination flip with respect to unstable manifold (e.g., 1D stable manifold).
- $i = 15$: Non-central homoclinic to saddle-node (in stable manifold).
- $i = 16$: Non-central homoclinic to saddle-node (in unstable manifold).

Expert users may wish to add their own test functions by editing the function **PSIHO** in **autlib5.f**.

It is important to remember that, in order to specify activated test functions, it is required to also add the corresponding label +20 to the list of continuation parameters and a zero of this parameter to the list of user-defined output points. Having done this, the corresponding parameters are output to the screen and zeros are accurately located.

20.7 Starting Strategies.

There are four possible starting procedures for continuation.

- (i) Data can be read from a previously-obtained output point from AUTO (e.g., from continuation of a periodic orbit up to large period; note that if the end-point of the data stored is not close to the equilibrium, a phase shift must be performed by setting **ISTART=4**). These data can be read from **fort.8** (saved to **s.xxx**) by making **IRS** correspond to the label of the data point in question.

- (ii) Data from numerical integration (e.g., computation of a stable periodic orbit, or an approximate homoclinic obtained by shooting) can be read in from a data file using the general AUTO utility `@fc` or `us` (see earlier in the manual). The numerical data should be stored in a file `xxx.dat`, in multi-column format according to the read statement

```
READ(...,*) T(J),(U(I,J),I=1,NDIM)
```

where T runs in the interval $[0,1]$. After running `@fc` or `us` the restart data is stored in the format of a previously computed solution in `s.dat`. When starting from this solution `IRS` should be set to 1 and the value of `ISTART` is irrelevant.

- (iii) By setting `ISTART=2`, an explicit homoclinic solution can be specified in the routine **STPNT** in the usual AUTO format, that is $U = \dots(T)$ where T is scaled to lie in the interval $[0,1]$.
- (iv) The choice `ISTART=3`, allows for a homotopy method to be used to approach a homoclinic orbit starting from a small approximation to a solution to the linear problem in the unstable manifold (Doedel, Friedman & Monteiro 1993). For details of implementation, the reader is referred to Section 5.1.2. of Champneys & Kuznetsov (1994), under the simplification that we do not solve for the adjoint $u(t)$ here. The basic idea is to start with a small solution in the unstable manifold, and perform continuation in `PAR(11)=2T` and dummy initial-condition parameters ξ_i in order to satisfy the correct right-hand boundary conditions, which are defined by zeros of other dummy parameters ω_i . More precisely, the left-hand end point is placed in the tangent space to the unstable manifold of the saddle and is characterized by `NUNSTAB` coordinates ξ_i satisfying the condition

$$\xi_1^2 + \xi_2^2 + \dots + \xi_{\text{NUNSTAB}}^2 = \epsilon_0^2,$$

where ϵ_0 is a user-defined small number. At the right-hand end point, `NUNSTUB` values ω_i measure the deviation of this point from the tangent space to the stable manifold of the saddle.

Suppose that `IEQUIB=0,1` and set `IP=12+IEQUIB*NDIM`. Then

```
PAR(IP)           :   $\epsilon_0$ 
PAR(IP+i)         :   $\xi_i, \quad i=1,2,\dots,\text{NUNSTAB}$ 
PAR(IP+NUNSTAB+i) :   $\omega_i, \quad i=1,2,\dots,\text{NUNSTAB}$ 
```

*Note that to avoid interference with the test functions (i.e. `PAR(21)-PAR(36)`), one must have `IP+2*NUNSTAB < 21`.*

If an ω_i is vanished, it can be frozen while another dummy or system parameter is allowed to vary in order to make consequently all $\omega_i = 0$. The resulting final solution gives the initial homoclinic orbit provided the right-hand end point is sufficiently close to the saddle. See Chapter 23 for an example, however, we recommend the homotopy method only for “expert users”.

To compute the orientation of a homoclinic orbit (in order to detect inclination-flip bifurcations) it is necessary to compute, in tandem, a solution to the modified adjoint variational equation, by setting `ITWIST=1`. In order to obtain starting data for such a computation when restarting from a point where just the homoclinic is computed, upon increasing `ITWIST` to 1, AUTO generates trivial data for the adjoint. Because the adjoint equations are linear, only a single step of Newton's method is required to enable these trivial data to converge to the correct unique bounded solution. This can be achieved by making a single continuation step in a trivial parameter (i.e. a parameter that does not appear in the problem).

Decreasing `ITWIST` to 0 automatically deletes the data for the adjoint from the continuation problem.

20.8 Notes on Running HomCont Demos.

HomCont demos are given in the following chapters. To copy all files of a demo `xxx` (for example, `san`), move to a clean directory and type `demo('xxx')`. Simply typing `make` or `make all` will then automatically execute all runs of the demo. At each step, the user is encouraged to plot the data saved by using the command `plot` (e.g., `plot('1')` plots the data saved in `b.1` and `s.1`).

Of course, in a real application, the runs will not have been prepared in advance, and AUTO-commands must be used. Such commands can be found in a table at the end of each chapter. A sequence of detailed AUTO -commands will be given in these tables as illustrated in Table 20.1 and Table 20.2 for two representative runs of HomCont demo `san`.

The user is encouraged to copy the format of one of these demos when constructing new examples.

The output of the HomCont demos reproduced in the following chapters is somewhat machine dependent, as already noted in Section 12.4. In exceptional circumstances, AUTO may reach its maximum number of steps `NMX` before a certain output point, or the label of an output point may change. In such case the user may have to make appropriate changes in the AUTO constants-files.

COMMAND	ACTION
<code>ld('san')</code>	load the problem definition
<code>run(c='san.1',h='san.1')</code>	get the HomCont constants-file and run AUTO /HomCont
<code>sv('6')</code>	save output-files as <code>b.6</code> , <code>s.6</code> , <code>d.6</code>
<code>@H san 1</code>	
<code>@sv 6</code>	

Table 20.1: These two sets of AUTO-Commands are equivalent.

COMMAND	ACTION
<i>run(c='san.9',h='san.9',s='6')</i>	get the HomCont constants-file and run AUTO /HomCont; restart solution read from s.6
<i>ap('6')</i>	append output-files to b.6, s.6, d.6
<i>@H san 9 6</i> <i>@ap 6</i>	

Table 20.2: These two sets of AUTO-Commands are equivalent.

Chapter 21

HomCont Demo : san.

21.1 Sandstede's Model.

Consider the system (Sandstede 1995a)

$$\begin{aligned}\dot{x} &= a x + b y - a x^2 + (\tilde{\mu} - \alpha z) x (2 - 3x) \\ \dot{y} &= b x + a y - \frac{3}{2} b x^2 - \frac{3}{2} a x y - (\tilde{\mu} - \alpha z) 2 y \\ \dot{z} &= c z + \mu x + \gamma x y + \alpha \beta (x^2 (1 - x) - y^2)\end{aligned}\tag{21.1}$$

as given in the file `san.f`. Choosing the constants appearing in (21.1) appropriately allows for computing inclination and orbit flips as well as non-orientable resonant bifurcations, see (Sandstede 1995a) for details and proofs. The starting point for all calculations is $a = 0$, $b = 1$ where there exists an explicit solution given by

$$(x(t), y(t), z(t)) = \left(1 - \left(\frac{1 - e^t}{1 + e^t} \right)^2, 4e^t \frac{1 - e^t}{(1 + e^t)^3}, 0 \right).$$

This solution is specified in the routine `STPNT`.

21.2 Inclination Flip.

We start by copying the demo to the current work directory and running the first step

```
@dm san
make first
```

This computation starts from the analytic solution above with $a = 0$, $b = 1$, $c = -2$, $\alpha = 0$, $\beta = 1$ and $\gamma = \mu = \tilde{\mu} = 0$. The homoclinic solution is followed in the parameters $(a, \tilde{\mu}) = (\text{PAR}(1), \text{PAR}(8))$ up to $a = 0.25$. The output is summarised on the screen as

BR	PT	TY	LAB	PAR(1)	L2-NORM	PAR(8)
1	1	EP	1	0.000000E+00	4.000000E-01 ...	0.000000E+00
1	5	UZ	2	2.500000E-01	4.030545E-01 ...	-3.620329E-11
1	10	EP	3	7.384434E-01	4.339575E-01 ...	-9.038826E-09

and saved in more detail as `b.1`, `s.1` and `d.1`.

Next we want to add a solution to the adjoint equation to the solution obtained at $a = 0.25$. This is achieved by making the change `ITWIST = 1` saved in `h.san.2`, and `IRS = 2`, `NMX = 2` and `ICP(1) = 9` saved in `c.san.2`. We also disable any user-defined functions `NUZR=0`. The computation so-defined is a single step in a trivial parameter `PAR(9)` (namely a parameter that does not appear in the problem). The effect is to perform a Newton step to enable AUTO to converge to a solution of the adjoint equation.

`make second`

The output is stored in `b.2`, `s.2` and `d.2`.

We can now continue the homoclinic plus adjoint in $(\alpha, \tilde{\mu}) = (\text{PAR}(4), \text{PAR}(8))$ by changing the constants (stored in `c.san.3`) to read `IRS = 4`, `NMX = 50` and `ICP(1) = 4`. We also add `PAR(10)` to the list of continuation parameters `NICP, (ICP(I), I=1 NICP)`. Here `PAR(10)` is a dummy parameter used in order to make the continuation of the adjoint well posed. Theoretically, it should be zero if the computation of the adjoint is successful (Sandstede 1995a). The test functions for detecting resonant bifurcations (`ISPI(1)=1`) and inclination flips (`ISPI(1)=13`) are also activated. Recall that this should be specified in three ways. First we add `PAR(21)` and `PAR(33)` to the list of continuation parameters in `c.san.3`, second we set up user defined output at zeros of these parameters in the same file, and finally we set `NPSI=2` (`IPSI(1), IPSI(2))=1, 13` in `h.san.3`. We also add to `c.san.3` another user zero for detecting when `PAR(4)=1.0`. Running

`make third`

reads starting data from `s.2` and outputs to the screen

BR	PT	TY	LAB	PAR(4)	...	PAR(8)	PAR(10)	...	PAR(33)
1	20		5	7.847219E-01	...	-3.001440E-11	-4.268884E-09	...	-1.441124E+01
1	27	UZ	6	1.000000E+00	...	-3.844872E-11	-4.460769E-09	...	-5.701675E+00
1	35	UZ	7	1.230857E+00	...	-5.833977E-11	-4.530541E-09	...	9.434843E-06
1	40		8	1.383969E+00	...	-8.133899E-11	-4.671817E-09	...	1.348810E+00
1	50	EP	9	1.695209E+00	...	-1.386324E-10	-5.098460E-09	...	5.311065E-01

Full output is stored in `b.3`, `s.3` and `d.3`. Note that the artificial parameter $\epsilon = \text{PAR}(10)$ is zero within the allowed tolerance. At label 7, a zero of test function ψ_{13} has been detected which corresponds to an inclination flip with respect to the stable manifold. That the orientation of the homoclinic loop changes as the family passes through this point can be read from the information in `d.3`. However in `d.3`, the line

`ORIENTABLE (0.2982090775D-03)`

at `PT=35` would seem to contradict the detection of the inclination flip at this point. Nonetheless, the important fact is the zero of the test function; and note that the value of the variable indicating the orientation is small compared to its value at the other regular points. Data for the adjoint equation at `LAB= 5, 7` and `9` at and on either side of the inclination flip are presented in Fig. 21.1. The switching of the solution between components of the leading unstable left eigenvector is apparent. Finally, we remark that the Newton step in the dummy parameter `PAR(20)` performed above is crucial to obtain convergence. Indeed, if instead we try to continue the homoclinic orbit and the solution of the adjoint equation directly by setting

ITWIST = 1 IRS = 2 NMX = 50 ICP(1) = 4 NPUSZR = 0

(as saved in `c.san.4`) and running

`make fourth`

we obtain a no convergence error.

21.3 Non-orientable Resonant Eigenvalues.

Inspecting the output saved in the third run, we observe the existence of a non-orientable homoclinic orbit at label 7 corresponding to $N=40$. We restart at this label, with the first continuation parameter being once again $a = \text{PAR}(1)$, by changing constants and storing them in `c.san.5` according to

IRS = 7 DS = -0.05D0 NMX = 20 ICP(1) = 1

Running,

`make fifth`

the output at label 10

BR	PT	TY	LAB	PAR(1)	PAR(8)	PAR(10)	PAR(21)
1	8	UZ	10	-1.304570E-07	... 3.874816E-12	-1.468457E-09	-2.609139E-07

indicates that AUTO has detected a zero of $\text{PAR}(21)$, implying that a non-orientable resonant bifurcation occurred at that point.

21.4 Orbit Flip.

In this section we compute an orbit flip. To this end we restart from the original explicit solution, without computing the orientation. We begin by separately performing continuation in $(\alpha, \tilde{\mu})$, $(\beta, \tilde{\mu})$, $(a, \tilde{\mu})$, $(b, \tilde{\mu})$ and $(\mu, \tilde{\mu})$ in order to reach the parameter values $(a, b, \alpha, \beta, \mu) = (0.5, 3, 1, 0, 0.25)$. The sequence of continuations up to the desired parameter values are run via

make sixth
make seventh
make eighth
make ninth
make tenth

with appropriate continuation parameters and user output values set in the corresponding files `c.san.xx`. All the output is saved to `s.6`.

The final saved point `LAB=10` contains a homoclinic solution at the desired parameter values. From here we perform continuation in the negative direction of $(\mu, \tilde{\mu}) = (\text{PAR}(7), \text{PAR}(8))$ with the test function ψ_{11} for orbit flips with respect to the stable manifold activated.

make eleventh

The output detects an inclination flip (by a zero of `PAR(31)`) at `PAR(7)=0`

BR	PT	TY	LAB	PAR(7)	...	PAR(8)	PAR(31)
1	5	UZ	12	2.394737E-07	...	6.434492E-08	-4.133994E-06

at which parameter value the homoclinic orbit is contained in the (x, y) -plane (see Fig. 21.2).

Finally, we demonstrate that the orbit flip can be continued as three parameters (`PAR(6)`, `PAR(7)`, `PAR(8)`) are varied.

make twelfth

BR	PT	TY	LAB	PAR(7)	...	PAR(8)	PAR(6)
1	5		14	-5.374538E-19	...	-1.831991E-10	-3.250000E-01
1	10		15	-6.145911E-19	...	-2.628607E-10	-8.250001E-01
1	15		16	-4.947133E-19	...	-2.361151E-10	-1.325000E+00
1	20	EP	17	-5.792940E-19	...	-3.075527E-10	-1.825000E+00

The orbit flip continues to be defined by a planar homoclinic orbit at `PAR(7)=PAR(8)=0`.

21.5 Detailed AUTO -Commands.

AUTO -COMMAND	ACTION
<code>! mkdir san</code> <code>cd san</code> <code>demo('san')</code>	create an empty work directory change directory copy the demo files to the work directory
<code>run(c='san.1',h='san.1')</code> <code>sv('1')</code>	continuation in <code>PAR(1)</code> save output-files as <code>b.1, s.1, d.1</code>
<code>run(c='san.2',h='san.2',s='1')</code> <code>sv('2')</code>	generate adjoint variables; restart from <code>s.1</code> save output-files as <code>b.2, s.2, d.2</code>
<code>run(c='san.3',h='san.3',s='2')</code> <code>sv('3')</code>	continue homoclinic orbit and adjoint; restart from <code>s.2</code> save output-files as <code>b.3, s.3, d.3</code>
<code>run(c='san.4',h='san.4',s='1')</code> <code>sv('4')</code>	no convergence without dummy step; restart from <code>s.1</code> save output-files as <code>b.4, s.4, d.4</code>
<code>run(c='san.5',h='san.5',s='3')</code> <code>sv('5')</code>	continue non-orientable orbit; restart from <code>s.3</code> save output-files as <code>b.5, s.5, d.5</code>

Table 21.1: Detailed AUTO -Commands for running demo `san`.

AUTO -COMMAND	ACTION
<code>run(c='san.6',h='san.6',s='san')</code> <code>sv('6')</code>	restart and homotopy to <code>PAR(4)=1.0</code> save output-files as <code>b.6, s.6, d.6</code>
<code>run(c='san.7',h='san.7',s='6')</code> <code>ap('6')</code>	homotopy to <code>PAR(5)=0.0</code> ; restart from <code>s.6</code> append output-files to <code>b.6, s.6, d.6</code>
<code>run(c='san.8',h='san.8',s='6')</code> <code>ap('6')</code>	homotopy to <code>PAR(1)=0.5</code> ; restart from <code>s.6</code> append output-files to <code>b.6, s.6, d.6</code>
<code>run(c='san.9',h='san.9',s='6')</code> <code>ap('6')</code>	homotopy to <code>PAR(2)=3.0</code> ; restart from <code>s.6</code> append output-files to <code>b.6, s.6, d.6</code>
<code>run(c='san.10',h='san.10',s='6')</code> <code>ap('6')</code>	homotopy to <code>PAR(7)=0.25</code> ; restart from <code>s.6</code> append output-files to <code>b.6, s.6, d.6</code>
<code>run(c='san.11',h='san.11',s='6')</code> <code>sv('11')</code>	continue in <code>PAR(7)</code> to detect orbit flip; restart from <code>s.6</code> save output-files as <code>b.11, s.11, d.11</code>
<code>run(c='san.12',h='san.12',s='11')</code> <code>sv('12')</code>	three-parameter continuation of orbit flip; restart from <code>s.11</code> save output-files as <code>b.12, s.12, d.12</code>

Table 21.2: Detailed AUTO -Commands for running demo `san`.

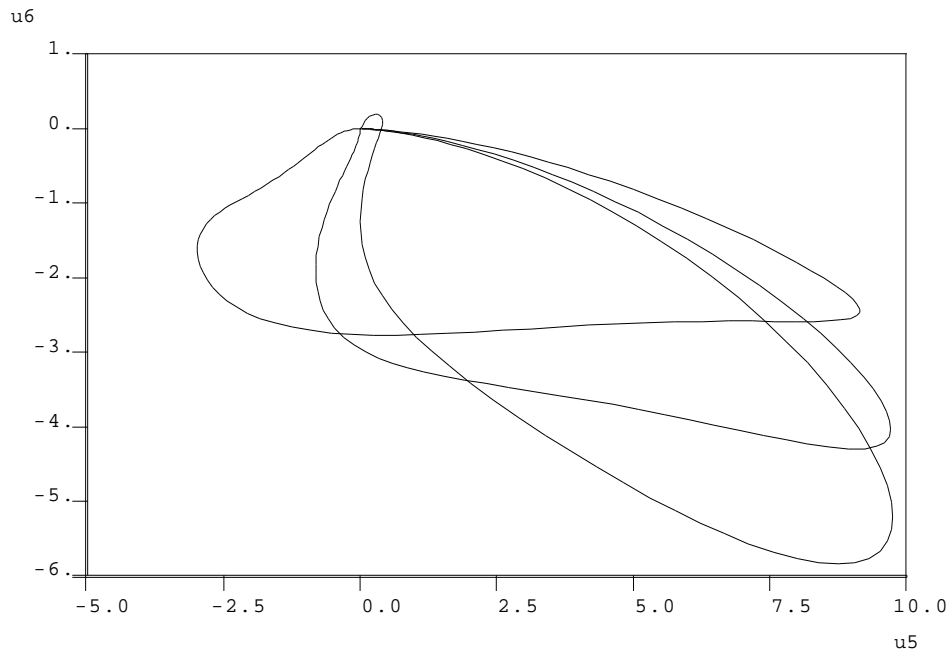


Figure 21.1: Second versus third component of the solution to the adjoint equation at labels 5, 7 and 9

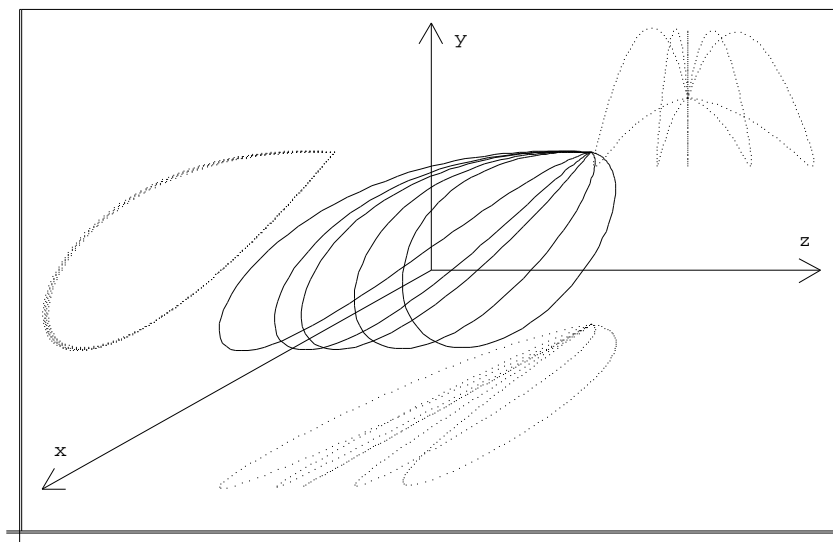


Figure 21.2: Orbits on either side of the orbit flip bifurcation. The critical orbit is contained in the (x, y) -plane

Chapter 22

HomCont Demo : mtn.

22.1 A Predator-Prey Model with Immigration.

Consider the following system of two equations (Scheffer 1995)

$$\begin{aligned}\dot{X} &= RX \left(1 - \frac{X}{K}\right) - \frac{A_1XY}{B_1 + X} + D_0K \\ \dot{Y} &= E_1 \frac{A_1XY}{B_1 + X} - D_1Y - \frac{A_2ZY^2}{B_2^2 + Y^2}.\end{aligned}\tag{22.1}$$

The values of all parameters except (K, Z) are set as follows :

$$R = 0.5, \quad A_1 = 0.4, \quad B_1 = 0.6, \quad D_0 = 0.01, \quad E_1 = 0.6, \quad A_2 = 1.0, \quad B_2 = 0.5, \quad D_1 = 0.15.$$

The parametric portrait of the system (22.1) on the (Z, K) -plane is presented in Figure 22.1. It contains fold $(t_{1,2})$ and Hopf (H) bifurcation curves, as well as a homoclinic bifurcation curve P . The fold curves meet at a cusp singular point C , while the Hopf and the homoclinic curves originate at a Bogdanov-Takens point BT . Only the homoclinic curve P will be considered here, the other bifurcation curves can be computed using AUTO or, for example, locbif (Khibnik, Kuznetsov, Levitin & Nikolaev 1993).

22.2 Continuation of Central Saddle-Node Homoclinics.

Local bifurcation analysis shows that at $K = 6.0$, $Z = 0.06729762\dots$, the system has a saddle-node equilibrium

$$(X^0, Y^0) = (5.738626\dots, 0.5108401\dots),$$

with one zero and one negative eigenvalue. Direct simulations reveal a homoclinic orbit to this saddle-node, departing and returning along its central direction (i.e., tangent to the null-vector).

Starting from this solution, stored in the file `mtn.dat`, we continue the saddle-node central homoclinic orbit with respect to the parameters K and Z by copying the demo and running it

*@dm mtn
make first*

The file `mtn.f` contains approximate parameter values

$$K = \text{PAR}(1) = 6.0, \quad Z = \text{PAR}(2) = 0.06729762,$$

as well as the coordinates of the saddle-node

$$X^0 = \text{PAR}(12) = 5.738626, \quad Y^0 = \text{PAR}(13) = 0.5108401,$$

and the length of the truncated time-interval

$$T_0 = \text{PAR}(11) = 1046.178.$$

Since a homoclinic orbit to a saddle-node is being followed, we have also made the choices

$$\text{IEQUIB} = 2 \quad \text{NUNSTAB} = 0 \quad \text{NSTAB} = 1$$

in `h.mtn.1`. The two test-functions, ψ_{15} and ψ_{16} , to detect non-central saddle-node homoclinic orbits are also activated, which must be specified in three ways. Firstly, in `h.mtn.1`, `NPSI` is set to 2 and the active test functions `IPSI(I), I=1,2` are chosen as 15 and 16. This sets up the monitoring of these test functions. Secondly, in `c.mtn.1` user-defined functions (`NUZR=2`) are set up to look for zeros of the parameters corresponding to these test functions. Recall that the parameters to be zeroed are always the test functions plus 20. Finally, these parameters are included in the list of continuation parameters (`NICP, (ICP(I), I=1 NICP)`).

Among the output there is a line

BR	PT	TY	LAB	PAR(1)	...	PAR(2)	PAR(35)	PAR(36)
1	27	UZ	5	6.10437E+00	...	6.932475E-02	-6.782898E-07	8.203437E-02

indicating that a zero of the test function `IPSI(1)=15` This means that at

$$D_1 = (K^1, Z^1) = (6.6104\dots, 0.069325\dots)$$

the homoclinic orbit to the saddle-node becomes *non-central*, namely, it returns to the equilibrium along the stable eigenvector, forming a non-smooth loop. The output is saved in `b.1`, `s.1` and `d.1`. Repeating computations in the opposite direction along the curve, `IRS=1`, `DS=-0.01` in `c.mtn.2`,

make second

one obtains

BR	PT	TY	LAB	PAR(1)	...	PAR(2)	PAR(35)	PAR(36)
1	34	UZ	9	5.180323E+00	...	6.385506E-02	3.349720E-09	9.361957E-02

which means another non-central saddle-node homoclinic bifurcation occurs at

$$D_2 = (K^2, Z^2) = (5.1803\dots, 0.063855\dots).$$

Note that these data were obtained using a smaller value of `NTST` than the original computation (compare `c.mtn.1` with `c.mtn.2`). The high original value of `NTST` was only necessary for the first few steps because the original solution is specified on a uniform mesh.

22.3 Switching between Saddle-Node and Saddle Homoclinic Orbits.

Now we can switch to continuation of saddle homoclinic orbits at the located codim 2 points D_1 and D_2 .

make third

starts from D_1 . Note that now

NUNSTAB = 1 IEQUIB = 1

has been specified in `h.mtn.3`. Also, test functions ψ_9 and ψ_{10} have been activated in order to monitor for non-hyperbolic equilibria along the homoclinic locus. We get the following output

BR	PT	TY	LAB	PAR(1)	...	PAR(2)	PAR(29)	PAR(30)
1	10		11	7.114523E+00	...	7.081751E-02	-4.649861E-01	3.183429E-03
1	20		12	9.176810E+00	...	7.678731E-02	-4.684912E-01	1.609294E-02
1	30		13	1.210834E+01	...	8.543468E-02	-4.718871E-01	3.069638E-02
1	40	EP	14	1.503788E+01	...	9.428036E-02	-4.743794E-01	4.144558E-02

The fact that PAR(29) and PAR(30) do not change sign indicates that there are no further non-hyperbolic equilibria along this family. Note that restarting in the opposite direction with IRS=11, DS=-0.02

make fourth

will detect the same codim 2 point D_1 but now as a zero of the test-function ψ_{10}

BR	PT	TY	LAB	PAR(1)	...	PAR(2)	PAR(29)	PAR(30)
1	10	UZ	15	6.610459E+00	...	6.932482E-02	-4.636603E-01	1.725013E-09

Note that the values of PAR(1) and PAR(2) differ from that at label 4 only in the sixth significant figure.

Actually, the program runs further and eventually computes the point D_2 and the whole lower family of P emanating from it, however, the solutions between D_1 and D_2 should be considered as spurious¹, therefore we do not save these data. The reliable way to compute the lower family of P is to restart computation of saddle homoclinic orbits in the other direction from the point D_2

make fifth

This gives the lower family of P approaching the BT point (see Figure 22.1)

¹ The program actually computes the saddle-saddle heteroclinic orbit bifurcating from the non-central saddle-node homoclinic at the point D_1 , see Champneys et al. (1996, Fig. 2), and continues it to the one emanating from D_2 .

BR	PT	TY	LAB	PAR(1)	...	PAR(2)	PAR(29)	PAR(30)
1	10		15	4.966429E+00	...	6.298418E-02	-4.382426E-01	4.946824E-03
1	20		16	4.925379E+00	...	7.961214E-02	-3.399102E-01	3.288447E-02
1	30		17	7.092267E+00	...	1.587114E-01	-1.692842E-01	3.876291E-02
1	40	EP	18	1.101819E+01	...	2.809825E-01	-3.482651E-02	2.104384E-02

The data are appended to the stored results in **b.1**, **s.1** and **d.1**. One could now display all data using the AUTO command `@p 1` to reproduce the curve P shown in Figure 22.1.

It is worthwhile to compare the homoclinic curves computed above with a curve $T_0 = \text{const}$ along which the system has a limit cycle of constant large period $T_0 = 1046.178$, which can easily be computed using AUTO or locbif. Such a curve is plotted in Figure 22.2. It obviously approximates well the saddle homoclinic loci of P , but demonstrates much bigger deviation from the saddle-node homoclinic segment D_1D_2 . This happens because the period of the limit cycle grows to infinity while approaching both types of homoclinic orbit, but with *different asymptotics*: as $-\ln \|\alpha - \alpha^*\|$, in the saddle homoclinic case, and as $\|\alpha - \alpha^*\|^{-1}$ in the saddle-node case.

22.4 Three-Parameter Continuation.

Finally, we can follow the curve of non-central saddle-node homoclinic orbits in three parameters. The extra continuation parameter is $D_0 = \text{PAR}(3)$. To achieve this we restart at label 4, corresponding to the codim 2 point D_1 . We return to continuation of saddle-node homoclinics, **NUNSTAB=0**, **IEQUIB=2**, but append the defining equation $\psi_{15} = 0$ to the continuation problem (via **NFIXED=1**, **IFIXED(1)=15**). The new continuation problem is specified in **c.mtn.6** and **h.mtn.6**.

make sixth

Notice that we set **ILP=1** and choose **PAR(3)** as the first continuation parameter so that AUTO can detect limit points with respect to this parameter. We also make a user-defined function (**NUZR=1**) to detect intersections with the plane $D_0 = 0.01$. We get among other output

BR	PT	TY	LAB	PAR(3)	L2-NORM	...	PAR(1)	PAR(2)
1	22	LP	19	1.081212E-02	5.325894E+00	...	5.673631E+00	6.608184E-02
1	31	UZ	20	1.000000E-02	4.819681E+00	...	5.180317E+00	6.385503E-02

the first line of which represents the D_0 value at which the homoclinic curve P has a tangency with the family t_2 of fold bifurcations. Beyond this value of D_0 , P consists entirely of saddle homoclinic orbits. The data at label 20 reproduce the coordinates of the point D_2 . The results of this computation and a similar one starting from D_1 in the opposite direction (with **DS=-0.01**) are displayed in Figure 22.3.

22.5 Detailed AUTO -Commands.

AUTO -COMMAND	ACTION
<i>! mkdir mtn</i>	create an empty work directory
<i>cd mtn</i>	change directory
<i>demo('mtn')</i>	copy the demo files to the work directory
<i>us('mtn')</i>	use the starting data in mtn.dat to create s.dat
<i>run(c='mtn.1',h='mtn.1',s='dat')</i> <i>sv('1')</i>	continue saddle-node homoclinic orbit save output-files as b.1 , s.1 , d.1
<i>run(c='mtn.2',h='mtn.2',s='1')</i> <i>ap('1')</i>	continue in opposite direction; restart from s.1 append output-files to b.1 , s.1 , d.1
<i>run(c='mtn.3',h='mtn.3',s='1')</i> <i>ap('1')</i>	switch to saddle homoclinic orbit ; restart from s.1 append output-files to b.1 , s.1 , d.1
<i>run(c='mtn.4',h='mtn.4',s='1')</i> <i>sv('4')</i>	continue in reverse direction; restart from s.1 save output-files as b.4 , s.4 , d.4
<i>run(c='mtn.5',h='mtn.5',s='1')</i> <i>ap('1')</i>	other saddle homoclinic orbit family; restart from s.1 append output-files to b. , s.1 , d.1
<i>run(c='mtn.6',h='mtn.6',s='1')</i> <i>sv('6')</i>	3-parameter non-central saddle-node homoclinic. save output-files as b.6 , s.6 , d.6

Table 22.1: Detailed AUTO -Commands for running demo **mtn**.

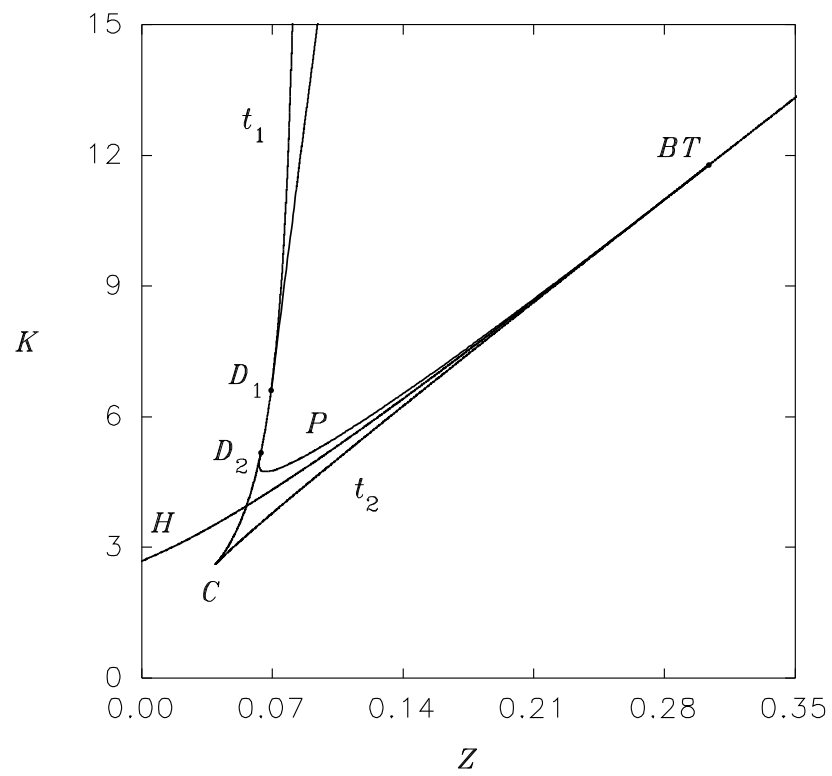


Figure 22.1: Parametric portrait of the predator-prey system

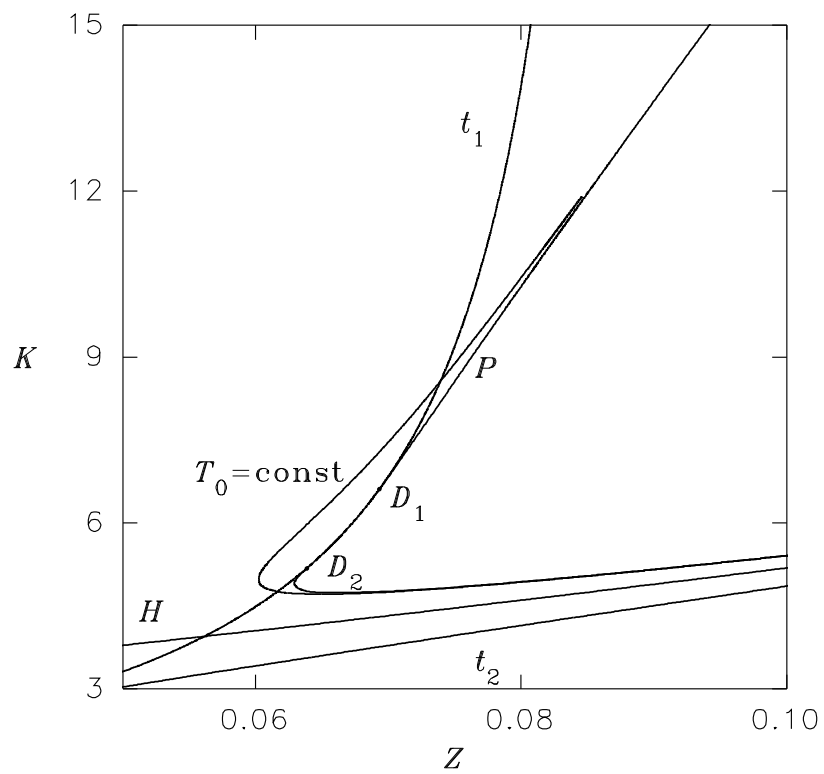


Figure 22.2: Approximation by a large-period cycle

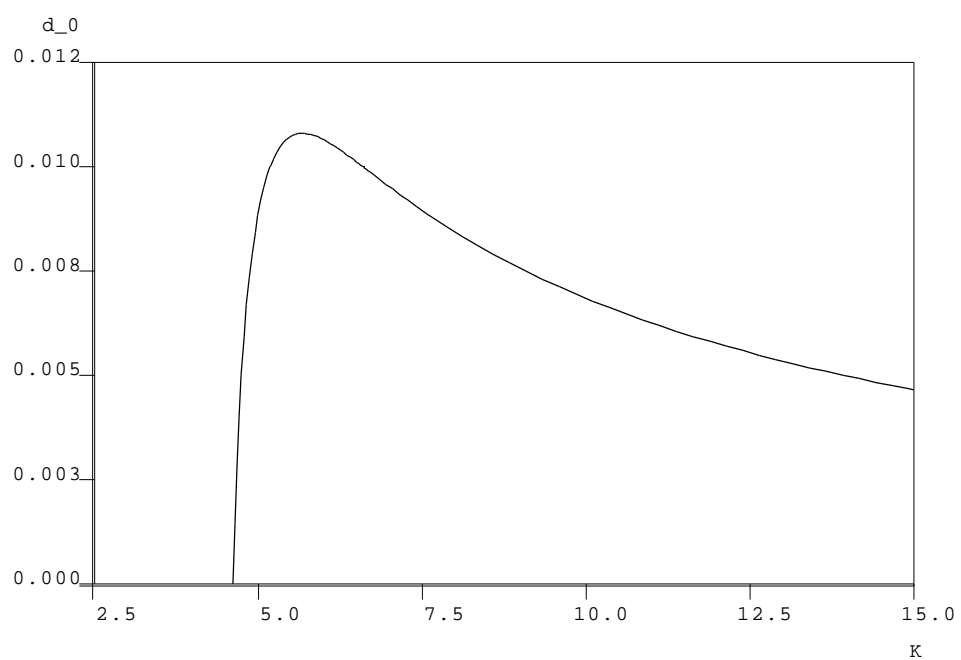


Figure 22.3: Projection onto the (K, D_0) -plane of the three-parameter curve of non-central saddle-node homoclinic orbit

Chapter 23

HomCont Demo : kpr.

23.1 Koper's Extended Van der Pol Model.

The equation-file `kpr.f` contains the equations

$$\begin{aligned}\dot{x} &= \epsilon_1^{-1} (k y - x^3 + 3 x - \lambda) \\ \dot{y} &= x - 2 y + z \\ \dot{z} &= \epsilon_2(y - z),\end{aligned}\tag{23.1}$$

with $\epsilon_1 = 0.1$ and $\epsilon_2 = 1$ (Koper 1995).

To copy across the demo `kpr` and compile we type

```
@dm kpr
```

23.2 The Primary Branch of Homoclinics.

First, we locate a homoclinic orbit using the homotopy method. The file `kpr.f` already contains approximate parameter values for a homoclinic orbit, namely $\lambda = \text{PAR}(1)=-1.851185$, $k = \text{PAR}(2)=-0.15$. The files `c.kpr.1` and `h.kpr.1` specify the appropriate constants for continuation in $2T = \text{PAR}(11)$ (also referred to as `PERIOD`) and the dummy parameter $\omega_1 = \text{PAR}(17)$ starting from a small solution in the local unstable manifold;

```
make first
```

Among the output there is the line

BR	PT	TY	LAB	PERIOD	L2-NORM	...	PAR(17)	...
1	29	UZ	2	1.900184E+01	1.693817E+00	...	4.433433E-09	...

which indicates that a zero of the artificial parameter ω_1 has been located. This means that the right-hand end point of the solution belongs to the plane that is tangent to the stable manifold at the saddle. The output is stored in files `b.1`, `s.1`, `d.1`. Upon plotting the data at label `2` (see Figure 23.1) it can be noted that although the right-hand projection boundary condition is satisfied, the solution is still quite away from the equilibrium.

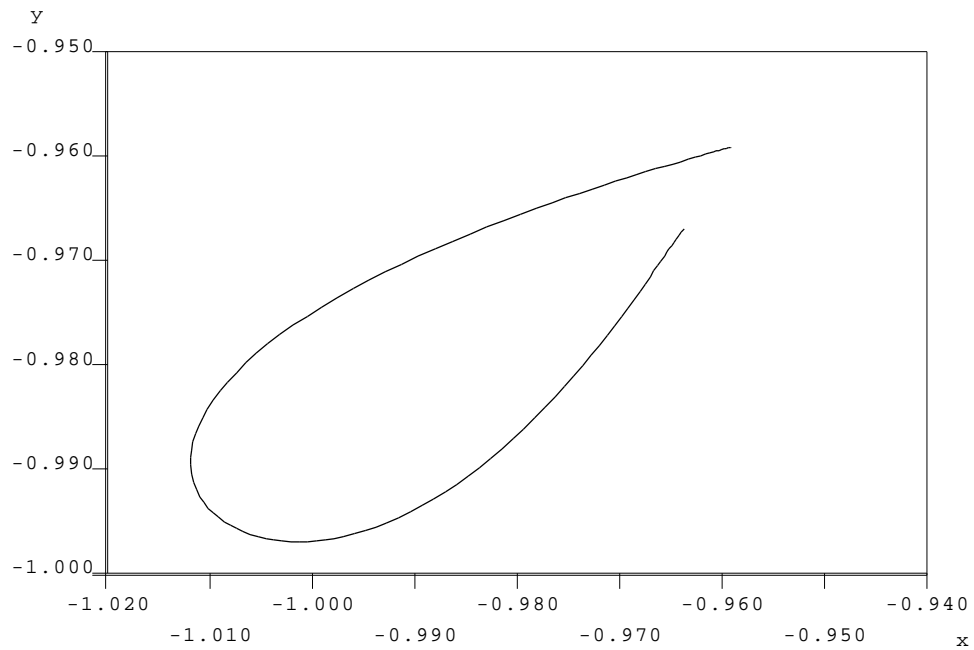


Figure 23.1: Projection on the (x, y) -plane of solutions of the boundary value problem with $2T = 19.08778$.

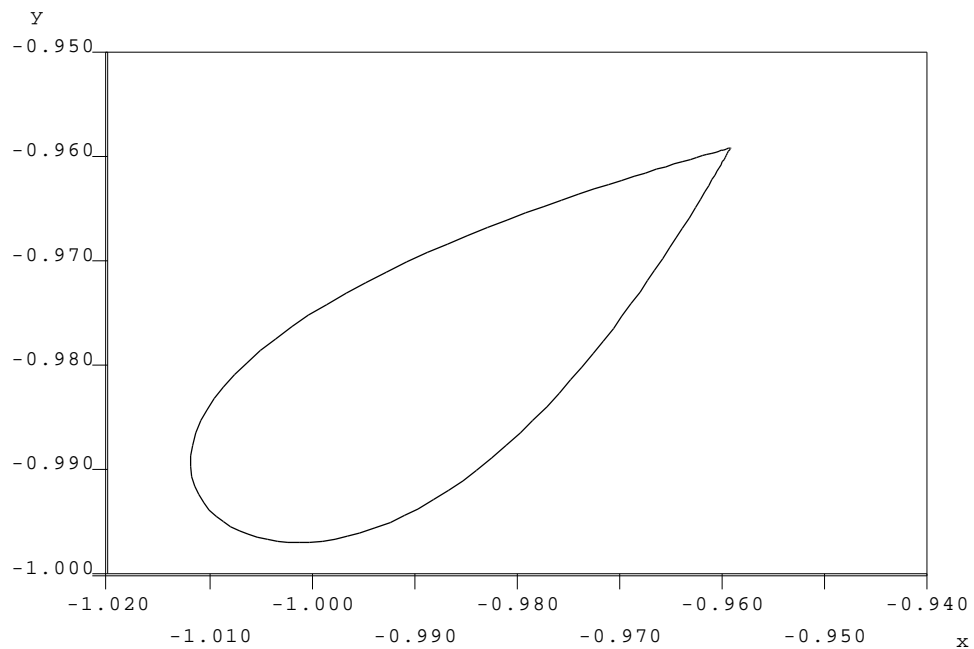


Figure 23.2: Projection on the (x, y) -plane of solutions of the boundary value problem with $2T = 60.0$.

The right-hand endpoint can be made to approach the equilibrium by performing a further continuation in T with the right-hand projection condition satisfied (`PAR(17)` fixed) but with λ allowed to vary.

`make second`

the output at label 4, stored in `kpr.2`,

BR	PT	TY	LAB	PERIOD	L2-NORM	...	PAR(1)	...
1	35	UZ	4	6.000000E+01	1.672806E+00	...	-1.851185E+00	...

provides a good approximation to a homoclinic solution (see Figure 23.2).

The second stage to obtain a starting solution is to add a solution to the modified adjoint variational equation. This is achieved by setting both `ITWIST` and `ISTART` to 1 (in `h.kpr.3`), which generates a trivial guess for the adjoint equations. Because the adjoint equations are linear, only a single Newton step (by continuation in a trivial parameter) is required to provide a solution. Rather than choose a parameter that might be used internally by `AUTO`, in `c.kpr.3` we take the continuation parameter to be `PAR(11)`, which is not quite a trivial parameter but whose affect upon the solution is mild.

`make third`

The output at the second point (label 6) contains the converged homoclinic solution (variables (`U(1)`, `U(2)`, `U(3)`)) and the adjoint (`U(4)`, `U(5)`, `U(6)`)). We now have a starting solution and are ready to perform two-parameter continuation.

The fourth run

`make fourth`

continues the homoclinic orbit in `PAR(1)` and `PAR(2)`. Note that several other parameters appear in the output. `PAR(10)` is a dummy parameter that should be zero when the adjoint is being computed correctly; `PAR(29)`, `PAR(30)`, `PAR(33)` correspond to the test functions ψ_9, ψ_{10} and ψ_{13} . That these test functions were activated is specified in three places in `c.kpr.4` and `h.kpr.4` as described in Section 20.6.

Note that at the end-point of the family (reached when after `NMX=50` steps) `PAR(29)` is approximately zero which corresponds to a zero of ψ_9 , a non-central saddle-node homoclinic orbit. We shall return to the computation of this codimension-two point later. Before reaching this point, among the output we find two zeroes of `PAR(33)` (test function ψ_{13}) which gives the accurate location of two inclination-flip bifurcations,

BR	PT	TY	LAB	PAR(1)	...	PAR(2)	PAR(10)	...	PAR(33)
1	6	UZ	10	-1.801662E+00	...	-2.002660E-01	-7.255434E-07	...	-1.425714E-04
1	12	UZ	11	-1.568756E+00	...	-4.395468E-01	-2.156353E-07	...	4.514073E-07

That the test function really does have a regular zero at this point can be checked from the data saved in `b.3`, plotting `PAR(33)` as a function of `PAR(1)` or `PAR(2)`. Figure 23.3 presents solutions $\phi(t)$ of the modified adjoint variational equation (for details see Champneys et al. (1996)) at parameter values on the homoclinic family before and after the first detected inclination

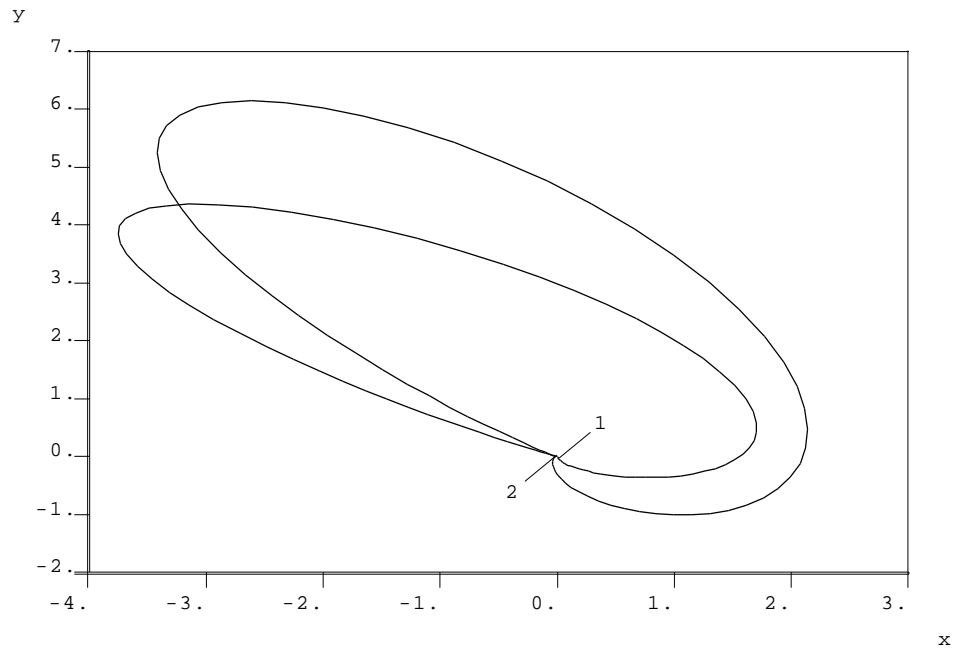


Figure 23.3: Projection on the (x, y) -plane of solutions $\phi(t)$ at 1 ($\lambda = -1.825470, k = -0.1760749$) and 2 ($\lambda = -1.686154, k = -0.3183548$).

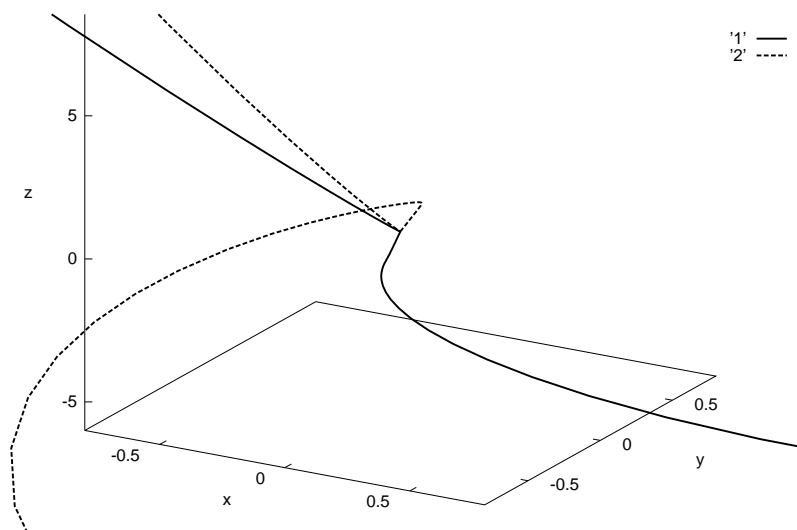


Figure 23.4: Three-dimensional blow-up of the solution curves $\phi(t)$ at labels 1 (dotted) and 2 (solid line) from Figure 3.8.

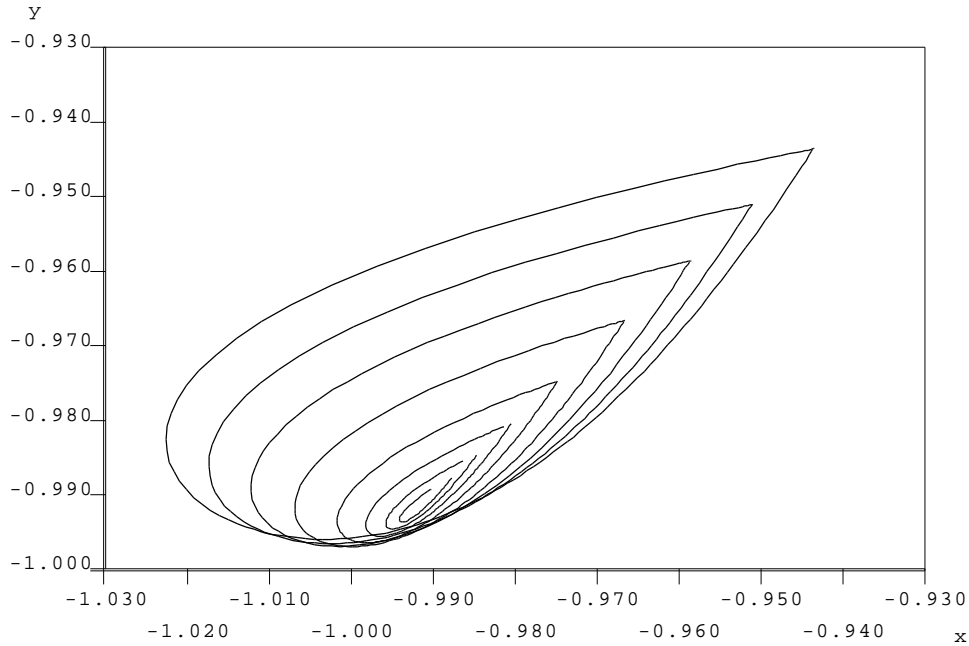


Figure 23.5: Computed homoclinic orbits approaching the BT point

flip. Note that these solutions were obtained by choosing a smaller step `DS` and more output (smaller `NPR`) in `c.kpr.4`. A blow-up of the region close to the origin of this figure is shown in Figure 23.4. It illustrates the flip of the solutions of the adjoint equation while moving through the bifurcation point. Note that the data in this figure were plotted after first performing an additional continuation of the solutions with respect to `PAR(11)`.

Continuing in the other direction

`make fifth`

we approach a Bogdanov-Takens point

BR	PT	TY	LAB	PAR(1)	...	PAR(10)	...	PAR(33)
1	50	EP	13	-1.938276E+00	...	-7.523344E+00	...	6.310810E+01

Note that the numerical approximation has ceased to become reliable, since `PAR(10)` has now become large. Phase portraits of homoclinic orbits between the BT point and the first inclination flip are depicted in Figure 23.5. Note how the computed homoclinic orbits approaching the BT point have their endpoints well away from the equilibrium. To follow the homoclinic orbit to the BT point with more precision, we would need to first perform continuation in T (`PAR(11)`) to obtain a more accurate homoclinic solution.

23.3 More Accuracy and Saddle-Node Homoclinic Orbits.

Continuation in T in order to obtain an approximation of the homoclinic orbit over a longer interval is necessary for parameter values near a non-hyperbolic equilibrium (either a saddle-

node or BT) where the convergence to the equilibrium is slower. First, we start from the original homoclinic orbit computed via the homotopy method, label 4, which is well away from the non-hyperbolic equilibrium. Also, we shall no longer be interested in inclination flips so we set ITWIST=0 in c.kpr.6, and in order to compute up to PAR(11)=1000, we set up a user-defined function for this. Running AUTO with PAR(11) and PAR(2) as free parameters

make sixth

we obtain among the output

BR	PT	TY	LAB	PERIOD	L2-NORM	...	PAR(2)
1	35	UZ	6	1.000000E+03	1.661910E+00	...	-1.500000E-01

We can now repeat the computation of the family of saddle homoclinic orbits in PAR(1) and PAR(2) from this point with the test functions ψ_9 and ψ_{10} for non-central saddle-node homoclinic orbits activated

make seventh

The saddle-node point is now detected at

BR	PT	TY	LAB	PAR(1)	...	PAR(2)	PAR(29)	PAR(30)
1	30	UZ	8	1.765003E-01	...	-2.405345E+00	2.743361E-06	2.309317E+01

which is stored in s.7. That PAR(29) (ψ_9) is zeroed shows that this is a non-central saddle-node connecting the centre manifold to the strong stable manifold. Note that all output beyond this point, although a well-posed solution to the boundary-value problem, is spurious in that it no longer represents a homoclinic orbit to a saddle equilibrium (see Champneys et al. (1996)). If we had chosen to, we could continue in the other direction in order to approach the BT point more accurately by reversing the sign of DS in c.kpr.7.

The files c.kpr.9 and h.kpr.9 contain the constants necessary for switching to continuation of the central saddle-node homoclinic curve in two parameters starting from the non-central saddle-node homoclinic orbit stored as label 8 in s.7.

make eighth

In this run we have activated the test functions for saddle to saddle-node transition points along curves of saddle homoclinic orbits (ψ_{15} and ψ_{16}). Among the output we find

BR	PT	TY	LAB	PAR(1)	...	PAR(2)	PAR(35)	PAR(36)
1	38	UZ	13	1.765274E-01	...	-2.405284E+00	9.705426E-03	-5.464784E-07

which corresponds to the family of homoclinic orbits leaving the locus of saddle-nodes in a second non-central saddle-node homoclinic bifurcation (a zero of ψ_{16}).

Note that the parameter values do not vary much between the two codimension-two non-central saddle-node points (labels 8 and 13). However, Figure 23.6 shows clearly that between the two codimension-two points the homoclinic orbit rotates between the two components of the 1D stable manifold, i.e. between the two boundaries of the center-stable manifold of the saddle node. The overall effect of this process is the transformation of a nearby “small” saddle

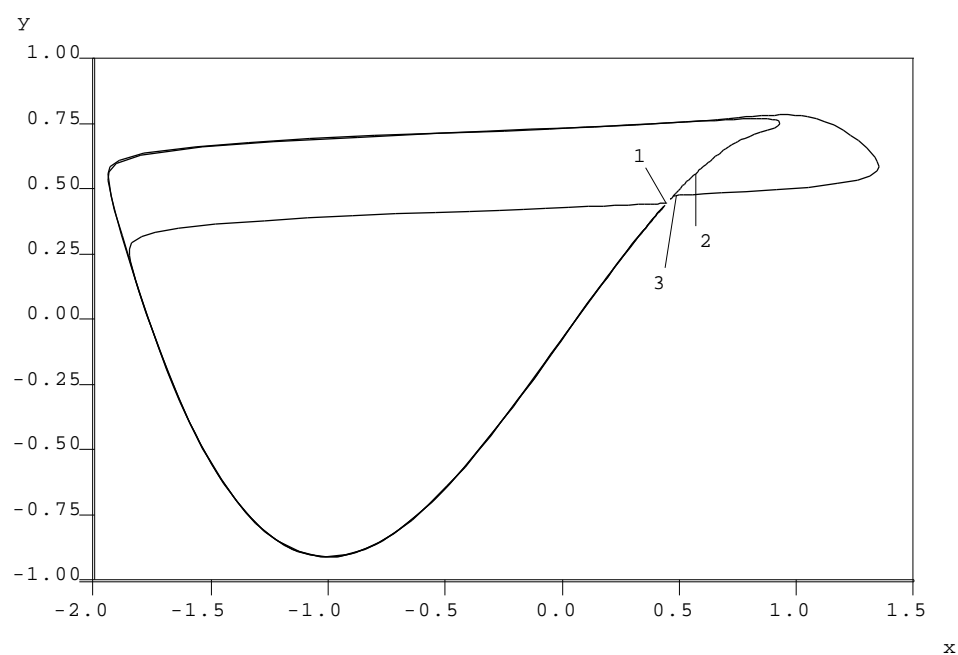


Figure 23.6: Two non-central saddle-node homoclinic orbits, 1 and 3; and, 2, a central saddle-node homoclinic orbit between these two points

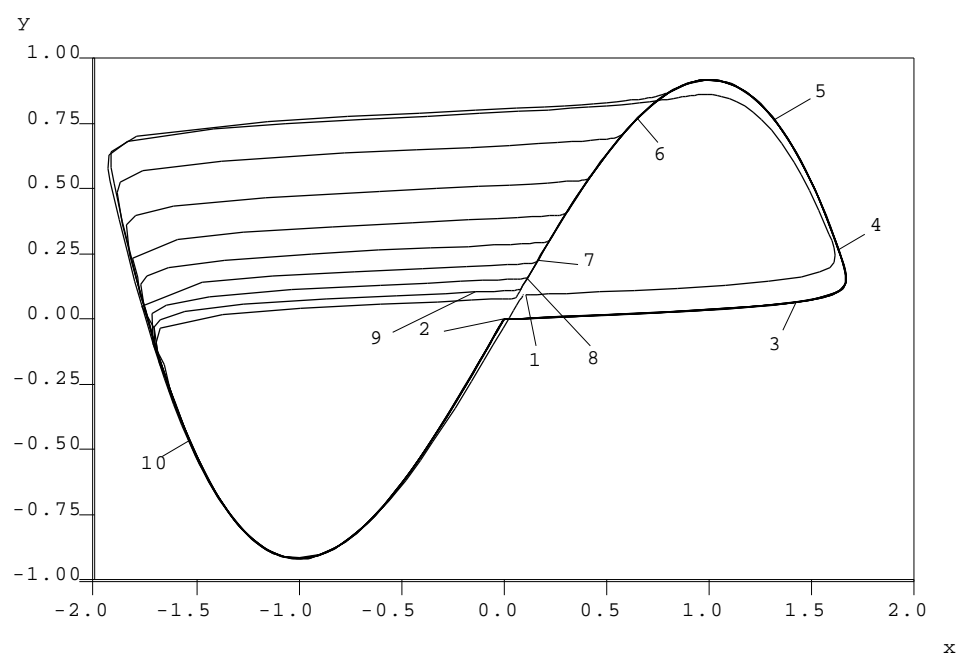


Figure 23.7: The big homoclinic orbit approaching a figure-of-eight

homoclinic orbit to a “big” saddle homoclinic orbit (i.e. with two extra turning points in phase space).

Finally, we can switch to continuation of the big saddle homoclinic orbit from the new codim 2 point at label 13.

make ninth

Note that AUTO takes a large number of steps near the line $\text{PAR}(1)=0$, while $\text{PAR}(2)$ approaches $-2.189\dots$ (which is why we chose such a large value $\text{NMX}=500$ in `c.kpr.9`). This particular computation ends at

BR	PT	TY	LAB	PAR(1)	L2-NORM	...	PAR(2)
1	500	EP	24	-1.218988E-05	2.181205E-01	...	-2.189666E+00

By plotting phase portraits of orbits approaching this end point (see Figure 23.7) we see a “canard-like” like transformation of the big homoclinic orbit to a pair of homoclinic orbits in a figure-of-eight configuration. That we get a figure-of-eight is not a surprise because $\text{PAR}(1)=0$ corresponds to a symmetry in the differential equations (Koper 1994); note also that the equilibrium, stored as ($\text{PAR}(12)$, $\text{PAR}(13)$, $\text{PAR}(14)$) in `d.9`, approaches the origin as we approach the figure-of-eight homoclinic.

23.4 Three-Parameter Continuation.

We now consider curves in three parameters of each of the codimension-two points encountered in this model, by freeing the parameter $\epsilon = \text{PAR}(3)$. First we continue the first inclination flip stored at label 7 in `s.3`

make tenth

Note that $\text{ITWIST}=1$ in `h.kpr.10`, so that the adjoint is also continued, and there is one fixed condition $\text{IFIXED}(1)=13$ so that test function ψ_{13} has been frozen. Among the output there is a codimension-three point (zero of ψ_9) where the neutrally twisted homoclinic orbit collides with the saddle-node curve

BR	PT	TY	LAB	PAR(1)	...	PAR(2)	PAR(3)	PAR(29)	...
1	28	UZ	14	1.282702E-01	...	-2.519325E+00	5.744770E-01	-4.347113E-09	...

The other detected inclination flip (at label 8 in `s.3`) is continued similarly

make eleventh

giving among its output another codim 3 saddle-node inclination-flip point

BR	PT	TY	LAB	PAR(1)	...	PAR(2)	PAR(3)	PAR(29)	...
1	27	UZ	14	1.535420E-01	...	-2.458100E+00	1.171705E+00	-1.933188E-07	...

Output beyond both of these codim 3 points is spurious and both computations end in an `MX` point (no convergence).

To continue the non-central saddle-node homoclinic orbits it is necessary to work on the data without the solution $\phi(t)$. We restart from the data saved at `LAB=8` and `LAB=13` in `s.7` and `s.8` respectively. We could continue these codim 2 points in two ways, either by appending the defining condition $\psi_{16} = 0$ to the continuation of saddle-node homoclinic orbits (with `IEQUIB=2`, etc.), or by appending $\psi_9 = 0$ to the continuation of a saddle homoclinic orbit (with `IEQUIB=1`. The first approach is used in the example `mtn`, for contrast we shall adopt the second approach here.

```
make twelfth
make thirteenth
```

The projection onto the (ϵ, k) -plane of all four of these codimension-two curves is given in Figure 23.8. The intersection of the inclination-flip lines with one of the non-central saddle-node homoclinic lines is apparent. Note that the two non-central saddle-node homoclinic orbit curves are almost overlaid, but that as in Figure 23.6 the orbits look quite distinct in phase space.

23.5 Detailed AUTO -Commands.

AUTO -COMMAND	ACTION
<code>! mkdir kpr</code> <code>cd kpr</code> <code>demo('kpr')</code>	create an empty work directory change directory copy the demo files to the work directory
<code>run(c='kpr.1',h='kpr.1')</code> <code>sv('1')</code>	continuation in the time-length parameter <code>PAR(11)</code> save output-files as <code>b.1, s.1, d.1</code>
<code>run(c='kpr.2',h='kpr.2',s='1')</code> <code>sv('2')</code>	locate the homoclinic orbit; restart from <code>s.1</code> save output-files as <code>b.2, s.2, d.2</code>
<code>run(c='kpr.3',h='kpr.3',s='2')</code> <code>sv('3')</code>	generate adjoint variables ; restart from <code>s.2</code> save output-files as <code>b.3, s.3, d.3</code>
<code>run(c='kpr.4',h='kpr.4',s='3')</code> <code>ap('3')</code>	continue the homoclinic orbit; restart from <code>s.3</code> append output-files to <code>b.3, s.3, d.3</code>
<code>run(c='kpr.5',h='kpr.5',s='3')</code> <code>ap('3')</code>	continue in reverse direction; restart from <code>s.3</code> append output-files to <code>b.3, s.3, d.3</code>
<code>run(c='kpr.6',h='kpr.6',s='2')</code> <code>sv('6')</code>	increase the period; restart from <code>s.2</code> save output-files as <code>b.6, s.6, d.6</code>

Table 23.1: Detailed AUTO -Commands for running demo `kpr`.

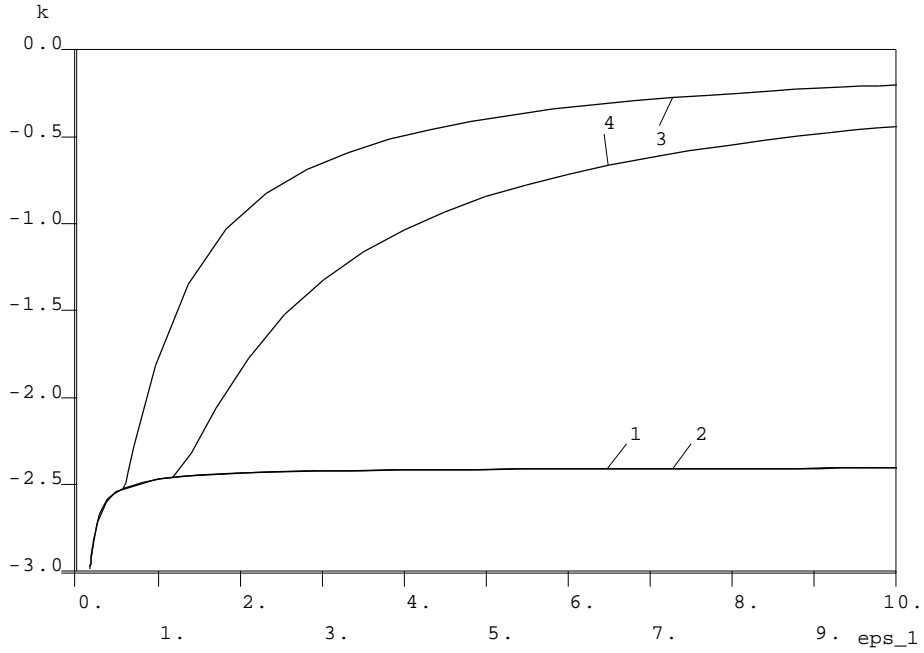


Figure 23.8: Projection onto the $(\text{PAR}(3), \text{PAR}(2))$ -plane of the non-central saddle-node homoclinic orbit curves (labeled 1 and 2) and the inclination-flip curves (labeled 3 and 4)

AUTO -COMMAND	ACTION
<code>run(c='kpr.7',h='kpr.7',s='6')</code> <code>sv('7')</code>	recompute the family of homoclinic orbits; restart from s.6 save output-files as b.7, s.7, d.7
<code>run(c='kpr.8',h='kpr.8',s='7')</code> <code>sv('8')</code>	continue central saddle-node homoclinics; restart from s.7 save output-files as b.8, s.8, d.8
<code>run(c='kpr.9',h='kpr.9',s='8')</code> <code>sv('9')</code>	continue homoclinics from codim-2 point; restart from s.8 save output-files as b.9, s.9, d.9
<code>run(c='kpr.10',h='kpr.10',s='3')</code> <code>sv('10')</code>	3-parameter curve of inclination-flips; restart from s.3 save output-files as b.10, s.10, d.10
<code>run(c='kpr.11',h='kpr.11',s='3')</code> <code>sv('11')</code>	another curve of inclination-flips; restart from s.3 save output-files as b.11, s.11, d.11
<code>run(c='kpr.12',h='kpr.12',s='7')</code> <code>sv('12')</code>	continue non-central saddle-node homoclinics; restart from s.7 save output-files as b.12, s.12, d.12
<code>run(c='kpr.13',h='kpr.13',s='8')</code> <code>ap('12')</code>	continue non-central saddle-node homoclinics; restart from s.8 append output-files to b.12, s.12, d.12

Table 23.2: Detailed AUTO -Commands for running demo kpr.

Chapter 24

HomCont Demo : cir.

24.1 Electronic Circuit of Freire *et al.*

Consider the following model of a three-variable electronic circuit (Freire, Rodríguez-Luis, Gamero & Ponce 1993)

$$\begin{cases} \dot{x} &= [-(\beta + \nu)x + \beta y - a_3 x^3 + b_3(y - x)^3] / r, \\ \dot{y} &= \beta x - (\beta + \gamma)y - z - b_3(y - x)^3, \\ \dot{z} &= y. \end{cases} \quad (24.1)$$

These autonomous equations are also considered in the AUTO demo `tor`.

First, we copy the demo into a new directory and compile

```
@dm cir
```

The system is contained in the equation-file `cir.f` and the initial run-time constants are stored in `c.cir.1` and `h.cir.1`. We begin by starting from the data from `cir.dat` for a saddle-focus homoclinic orbit at $\nu = -0.721309$, $\beta = 0.6$, $\gamma = 0$, $r = 0.6$, $A_3 = 0.328578$ and $B_3 = 0.933578$, which was obtained by shooting over the time interval $2T = \text{PAR}(11) = 36.13$. We wish to follow the family in the (β, ν) -plane, but first we perform continuation in (T, ν) to obtain a better approximation to a homoclinic orbit.

```
make first
```

yields the output

BR	PT	TY	LAB	PERIOD	L2-NORM	...	PAR(1)
1	21	UZ	2	1.000000E+02	1.286637E-01	...	-7.213093E-01
1	42	UZ	3	2.000000E+02	9.097899E-02	...	-7.213093E-01
1	50	EP	4	2.400000E+02	8.305208E-02	...	-7.213093E-01

Note that $\nu = \text{PAR}(1)$ remains constant during the continuation as the parameter values do not change, only the the length of the interval over which the approximate homoclinic solution is computed. Note from the eigenvalues, stored in `d.1` that this is a homoclinic orbit to a saddle-focus with a one-dimensional unstable manifold.

We now restart at LAB=3, corresponding to a time interval $2T = 200$, and change the principal continuation parameters to be (ν, β) . The new constants defining the continuation are given in c.cir.2 and h.cir.2. We also activate the test functions pertinent to codimension-two singularities which may be encountered along a family of saddle-focus homoclinic orbits, viz. $\psi_2, \psi_4, \psi_5, \psi_9$ and ψ_{10} . This must be specified in three ways: by choosing NPSI=5 and appropriate IPSI(I) in h.cir.2, by adding the corresponding parameter labels to the list of continuation parameters ICP(I) in c.cir.2 (recall that these parameter indices are 20 more than the corresponding ψ indices), and finally adding USZR functions defining zeros of these parameters in c.cir.2. Running

make second

results in

BR	PT	TY	LAB	PAR(1)	...	PAR(2)	...	PAR(25)	PAR(29)
1	17	UZ	5	-7.256925E-01	...	4.535645E-01	...	-1.765251E-05	-2.888436E-01
1	75	UZ	6	-1.014704E+00	...	9.998966E-03	...	1.664509E+00	-5.035997E-03
1	78	UZ	7	-1.026445E+00	...	-2.330391E-05	...	1.710804E+00	1.165176E-05
1	81	UZ	8	-1.038012E+00	...	-1.000144E-02	...	1.756690E+00	4.964621E-03
1	100	EP	9	-1.164160E+00	...	-1.087732E-01	...	2.230329E+00	5.042736E-02

with results saved in b.2, s.2, d.2. Upon inspection of the output, note that label 5, where $\text{PAR}(25) \approx 0$, corresponds to a neutrally-divergent saddle-focus, $\psi_5 = 0$. Label 7, where $\text{PAR}(29) \approx 0$ corresponds to a local bifurcation, $\psi_9 = 0$, which we note from the eigenvalues stored in d.2 corresponds to a *Shil'nikov-Hopf* bifurcation. Note that $\text{PAR}(2)$ is also approximately zero at label 7, which accords with the analytical observation that the origin of (24.1) undergoes a Hopf bifurcation when $\beta = 0$. Labels 6 and 8 are the user-defined output points, the solutions at which are plotted in Fig. 24.1. Note that solutions beyond label 7 (e.g., the plotted solution at label 8) do not correspond to homoclinic orbits, but to *point-to-cycle* heteroclinic orbits (c.f. Section 2.2.1 of Champneys et al. (1996)).

We now continue in the other direction along the family. It turns out that starting from the initial point in the other direction results in missing a codim 2 point which is close to the starting point. Instead we start from the first saved point from the previous computation (label 5 in s.2):

make third

The output

BR	PT	TY	LAB	PAR(1)	...	PAR(2)	PAR(22)	PAR(24)
1	9	UZ	10	-7.204001E-01	...	5.912315E-01	-1.725669E+00	-3.295862E-05
1	18	UZ	11	-7.590583E-01	...	7.428734E-01	3.432139E-05	-2.822988E-01
1	26	UZ	12	-7.746686E-01	...	7.746147E-01	5.833163E-01	1.637611E-07
1	28	EP	13	-7.746628E-01	...	7.746453E-01	5.908902E-01	1.426214E-04

contains a neutral saddle-focus (a *Belyakov* transition) at LAB=10 ($\psi_4 = 0$), a double real leading eigenvalue (saddle-focus to saddle transition) at LAB =11 ($\psi_2 = 0$) and a neutral saddle at LAB=12 ($\psi_4 = 0$). Data at several points on the complete family are plotted in Fig. 24.2. If

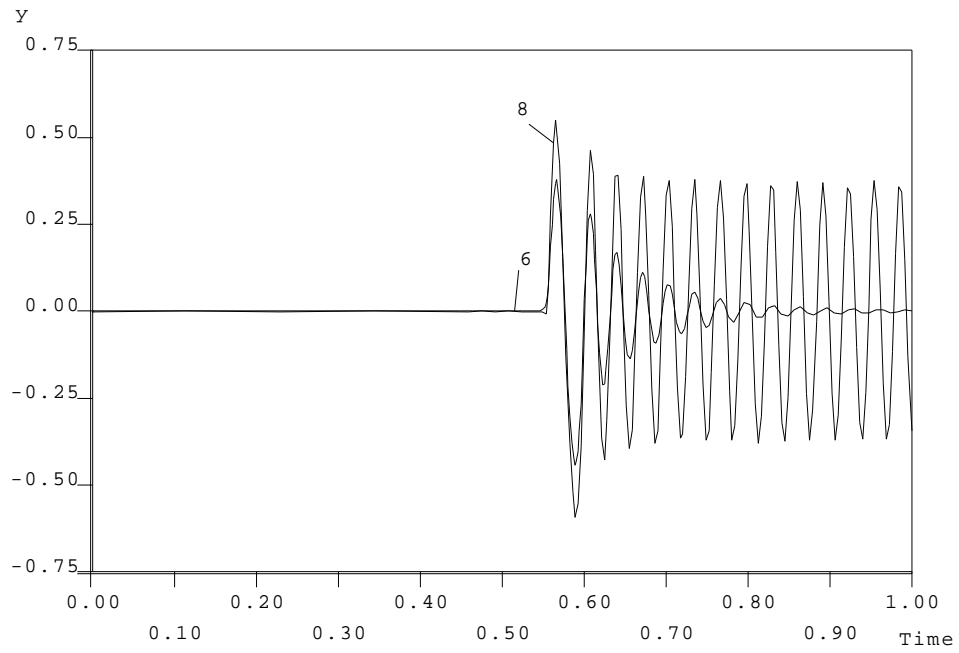


Figure 24.1: Solutions of the boundary value problem at labels 6 and 8, either side of the Shil'nikov-Hopf bifurcation

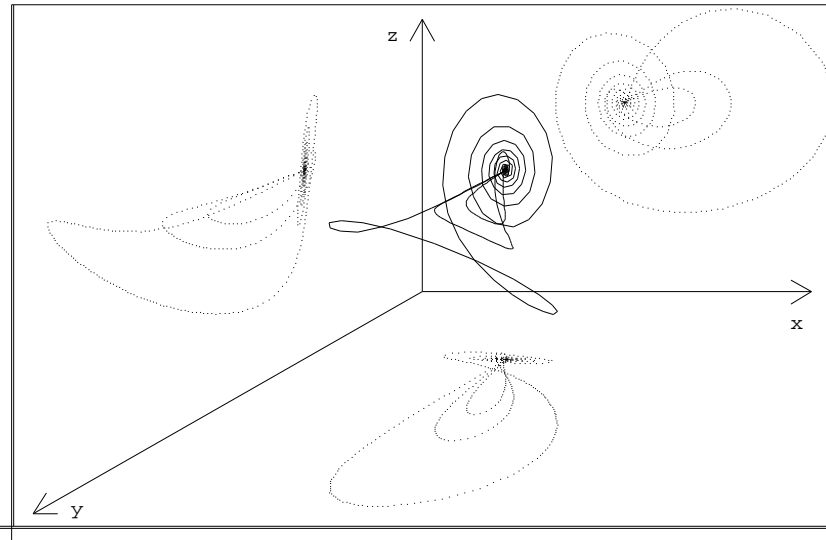


Figure 24.2: Phase portraits of three homoclinic orbits on the family, showing the saddle-focus to saddle transition

we had continued further (by increasing `MMX`), the computation would end at a no convergence error `TY=MX` owing to the homoclinic family approaching a Bogdanov-Takens singularity at small amplitude. To compute further towards the BT point we would first need to continue to a higher value of `PAR(11)`.

24.2 Detailed AUTO -Commands.

AUTO -COMMAND	ACTION
<code>! mkdir cir</code> <code>cd cir</code> <code>demo('cir')</code>	create an empty work directory change directory copy the demo files to the work directory
<code>us('cir')</code> <code>run(c='cir.1',h='cir.1',s='dat')</code> <code>sv('1')</code>	use the starting data in <code>cir.dat</code> to create <code>s.dat</code> increase the truncation interval; restart from <code>s.dat</code> save output-files as <code>b.1, s.1, d.1</code>
<code>run(c='cir.2',h='cir.2',s='1')</code> <code>sv('2')</code>	continue saddle-focus homoclinic orbit; restart from <code>s.1</code> save output-files as <code>b.2, s.2, d.2</code>
<code>run(c='cir.3',h='cir.3',s='2')</code> <code>ap('2')</code>	generate adjoint variables ; restart from <code>s.2</code> append output-files as <code>b.2, s.2, d.2</code>

Table 24.1: Detailed AUTO -Commands for running demo `cir`.

Chapter 25

HomCont Demo : she.

25.1 A Heteroclinic Example.

The following system of five equations Rucklidge & Mathews (1995)

$$\begin{aligned}
 \dot{x} &= \mu x + x y - z u, \\
 \dot{y} &= -y - x^2, \\
 \dot{z} &= (4\sigma x u + 4\sigma \mu z - 9\sigma z + 4x u + 4\mu z)/4(1 + \sigma) \\
 \dot{u} &= -\sigma u/4 - \sigma Q v/4\pi^2 + 3(1 + \sigma)xz/4\sigma \\
 \dot{v} &= \zeta u/4 - \zeta v/4
 \end{aligned} \tag{25.1}$$

has been used to describe shearing instabilities in fluid convection. The equations possess a rich structure of local and global bifurcations. Here we shall reproduce a single curve in the (σ, μ) -plane of codimension-one heteroclinic orbits connecting a non-trivial equilibrium to the origin for $Q = 0$ and $\zeta = 4$. The defining problem is contained in equation-file `she.f`¹, and starting data for the orbit at $(\sigma, \mu) = (0.5, 0.163875)$ are stored in `she.dat`, with a truncation interval of `PAR(11)=85.07`.

We begin by computing towards $\mu = 0$ with the option `IEQUIB=-2` which means that both equilibria are solved for as part of the continuation process.

```
@dm she
make first
```

This yields the output

BR	PT	TY	LAB	PAR(3)	L2-NORM	...	PAR(1)
1	5		2	4.528332E-01	3.726787E-01	...	1.364973E-01
1	10		3	3.943370E-01	3.303798E-01	...	1.044119E-01
1	15		4	3.358942E-01	2.873213E-01	...	7.515570E-02
1	20		5	2.772726E-01	2.433403E-01	...	4.952636E-02
1	25		6	2.181955E-01	1.981358E-01	...	2.845849E-02
1	30	EP	7	1.581633E-01	1.512340E-01	...	1.292975E-02

¹The last parameter used to store the equilibria (`PAR(21)`) is overlapped here with the first test-function. In this example, it is harmless since the test functions are irrelevant for heteroclinic continuation.

Alternatively, for this problem there exists an analytic expression for the two equilibria. This is specified in the subroutine **PVLS** of **she.f**. Re-running with **IEQUIB=-1**

make second

we obtain the output

BR	PT	TY	LAB	PAR(3)	L2-NORM	...	PAR(1)
1	5		2	4.432015E-01	3.657716E-01	...	1.310559E-01
1	10		3	3.723085E-01	3.142439E-01	...	9.300982E-02
1	15		4	3.008842E-01	2.611556E-01	...	5.933966E-02
1	20		5	2.286652E-01	2.062194E-01	...	3.179939E-02
1	25		6	1.555409E-01	1.491652E-01	...	1.239897E-02
1	30	EP	7	8.107462E-02	9.143108E-02	...	2.386616E-03

This output is similar to that above, but note that it is obtained slightly more efficiently because the extra parameters **PAR(12-21)** representing the coordinates of the equilibria are no longer part of the continuation problem. Also note that **AUTO** has chosen to take slightly larger steps along the family. Finally, we can continue in the opposite direction along the family from the original starting point (again with **IEQUIB=-1**).

make third

BR	PT	TY	LAB	PAR(3)	L2-NORM	...	PAR(1)
1	5		8	4.997590E-01	4.060153E-01	...	1.637322E-01
1	10		9	5.705299E-01	4.551872E-01	...	2.065264E-01
1	15		10	6.416439E-01	5.031844E-01	...	2.507829E-01
1	20		11	7.133301E-01	5.500668E-01	...	2.959336E-01
1	25		12	7.857688E-01	5.958712E-01	...	3.415492E-01
1	30		13	8.590970E-01	6.406182E-01	...	3.872997E-01
1	35	EP	14	9.334159E-01	6.843173E-01	...	4.329270E-01

The results of both computations are presented in Figure 25.1, from which we see that the orbit shrinks to zero as **PAR(1)= $\mu \rightarrow 0$** .

25.2 Detailed AUTO -Commands.

AUTO -COMMAND	ACTION
! mkdir she cd she demo('she')	create an empty work directory change directory copy the demo files to the work directory
us('she') run(c='she.1',h='she.1',s='dat') sv('1')	use the starting data in she.dat to create s.dat continue heteroclinic orbit; restart from s.dat save output-files as b.1, s.1, d.1
run(c='she.2',h='she.2',s='dat') sv('2')	repeat with IEQUIB=-1 save output-files as b.2, s.2, d.2
run(c='she.3',h='she.3',s='2') ap('2')	continue in reverse direction ; restart from s.2 append output-files to b.2, s.2, d.2

Table 25.1: Detailed AUTO -Commands for running demo she.

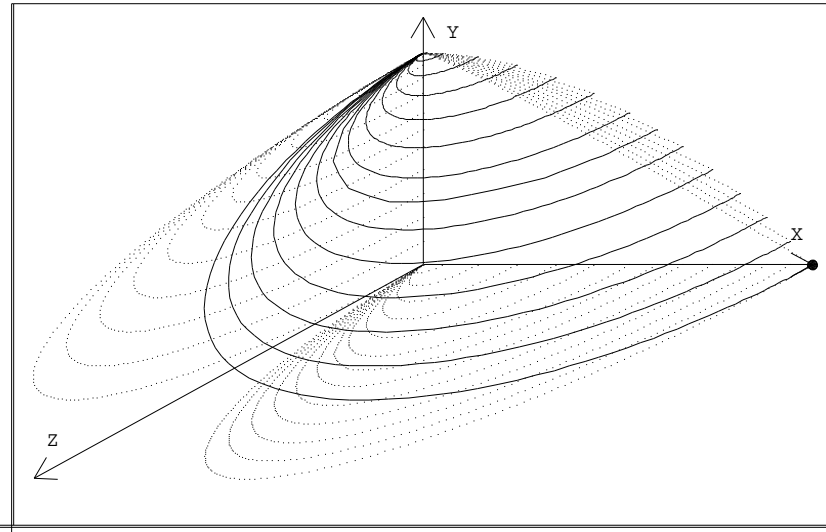


Figure 25.1: Projections into (x, y, z) -space of the family of heteroclinic orbits.

Chapter 26

HomCont Demo : rev.

26.1 A Reversible System.

The fourth-order differential equation

$$u'''' + Pu'' + u - u^3 = 0$$

arises in a number of contexts, e.g., as the travelling-wave equation for a nonlinear-Schrödinger equation with fourth-order dissipation (Buryak & Akhmediev 1995) and as a model of a strut on a symmetric nonlinear elastic foundation (Hunt, Bolt & Thompson 1989). It may be expressed as a system

$$\begin{cases} \dot{u}_1 = u_2 \\ \dot{u}_2 = u_3 \\ \dot{u}_3 = u_4 \\ \dot{u}_4 = -Pu_3 - u_1 + u_1^3 \end{cases} \quad (26.1)$$

Note that (26.1) is invariant under two separate reversibilities

$$R_1 : (u_1, u_2, u_3, u_4, t) \mapsto (u_1, -u_2, u_3, -u_4, -t) \quad (26.2)$$

and

$$R_2 : (u_1, u_2, u_3, u_4, t) \mapsto (-u_1, u_2, -u_3, u_4, -t) \quad (26.3)$$

First, we copy the demo into a new directory

```
@dm rev
```

For this example, we shall make two separate starts from data stored in equation and data files `rev.c.1`, `rev.dat.1` and `rev.c.3`, `rev.dat.3` respectively. The first of these contains initial data for a solution that is reversible under R_1 and the second for data that is reversible under R_2 .

26.2 An R_1 -Reversible Homoclinic Solution.

The first run

make first

starts by copying the files `rev.c.1` and `rev.dat.1` to `rev.f` and `rev.dat`. The orbit contained in the data file is a “primary” homoclinic solution for $P = 1.6$, with truncation (half-)interval $\text{PAR}(11) = 39.0448429$, which is reversible under R_1 . Note that this reversibility is specified in `h.rev.1` via `NREV=1, (IREV(I), I=1,NDIM) = 0 1 0 1`. Note also, from `c.rev.1` that we only have one free parameter `PAR(1)` because symmetric homoclinic orbits in reversible systems are generic rather than of codimension one. The first run results in the output

BR	PT	TY	LAB	PAR(1)	L2-NORM	MAX U(1)	...
1	7	UZ	2	1.700002E+00	2.633353E-01	4.179794E-01	
1	12	UZ	3	1.800000E+00	2.682659E-01	4.806063E-01	
1	15	UZ	4	1.900006E+00	2.493415E-01	4.429364E-01	
1	20	EP	5	1.996247E+00	1.111306E-01	1.007111E-01	

which is consistent with the theoretical result that the solution tends uniformly to zero as $P \rightarrow 0$. Note, by plotting the data saved in `s.1` that only “half” of the homoclinic orbit is computed up to its point of symmetry. See Figure 26.1.

The second run continues in the other direction of `PAR(1)`, with the test function ψ_2 activated for the detection of saddle to saddle-focus transition points

make second

The output

BR	PT	TY	LAB	PAR(1)	L2-NORM	MAX U(1)	...	PAR(22)
1	11	UZ	6	1.000005E+00	2.555446E-01	1.767149E-01	...	-3.000005E+00
1	22	UZ	7	-1.198325E-07	2.625491E-01	4.697314E-02	...	-2.000000E+00
1	33	UZ	8	-1.000000E+00	2.741483E-01	4.316007E-03	...	-1.000000E+00
1	44	UZ	9	-2.000000E+00	2.873838E-01	1.245735E-11	...	2.318248E-08
1	55	EP	10	-3.099341E+00	3.020172E-01	-2.749454E-11	...	1.099341E+00

shows a saddle to saddle-focus transition (indicated by a zero of `PAR(22)`) at `PAR(1)=-2`. Beyond that label the first component of the solution is negative and (up to the point of symmetry) monotone decreasing. See Figure 26.2.

26.3 An R_2 -Reversible Homoclinic Solution.

make third

Copies the files `rev.c.3` and `rev.dat.3` to `rev.f` and `rev.dat`, and runs them with the constants stored in `c.rev.3` and `h.rev.3`. The orbit contained in the data file is a “multi-pulse” homoclinic solution for $P = 1.6$, with truncation (half-)interval $\text{PAR}(11) = 47.4464189$, which is reversible under R_2 . This reversibility is specified in `h.rev.1` via `NREV=1, (IREV(I), I=1,NDIM) = 1 0 1 0`. The output

BR	PT	TY	LAB	PAR(1)	L2-NORM	MAX U(1)	...
1	15	UZ	2	1.700000E+00	3.836401E-01	4.890015E-01	
1	16	LP	3	1.711574E+00	3.922135E-01	5.442385E-01	

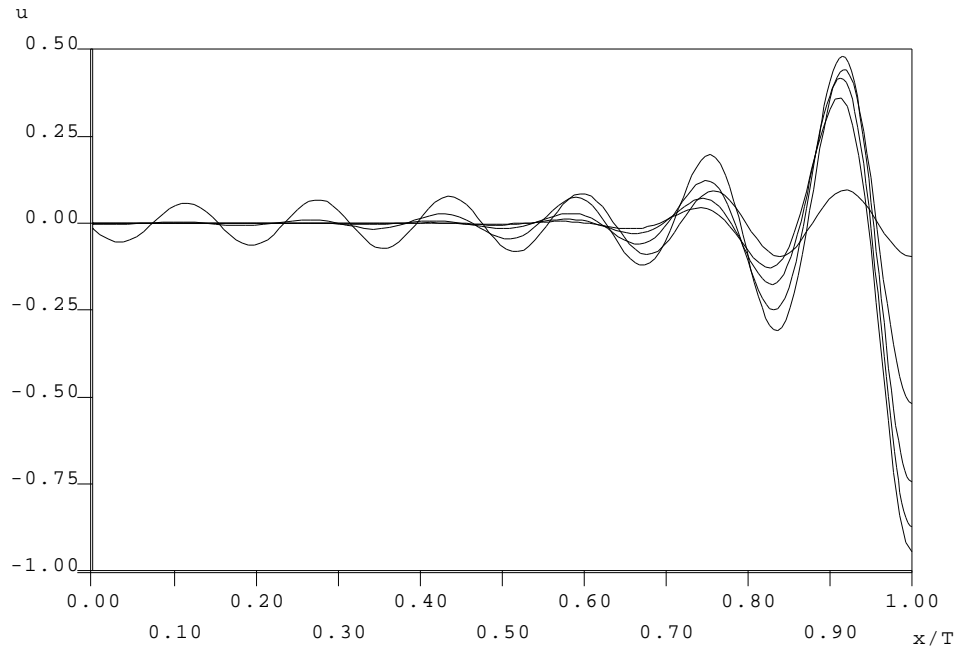


Figure 26.1: R_1 -Reversible homoclinic solutions on the half-interval $x/T \in [0, 1]$ where $T = 39.0448429$ for P approaching 2 (solutions with labels 1-5 respectively have decreasing amplitude)

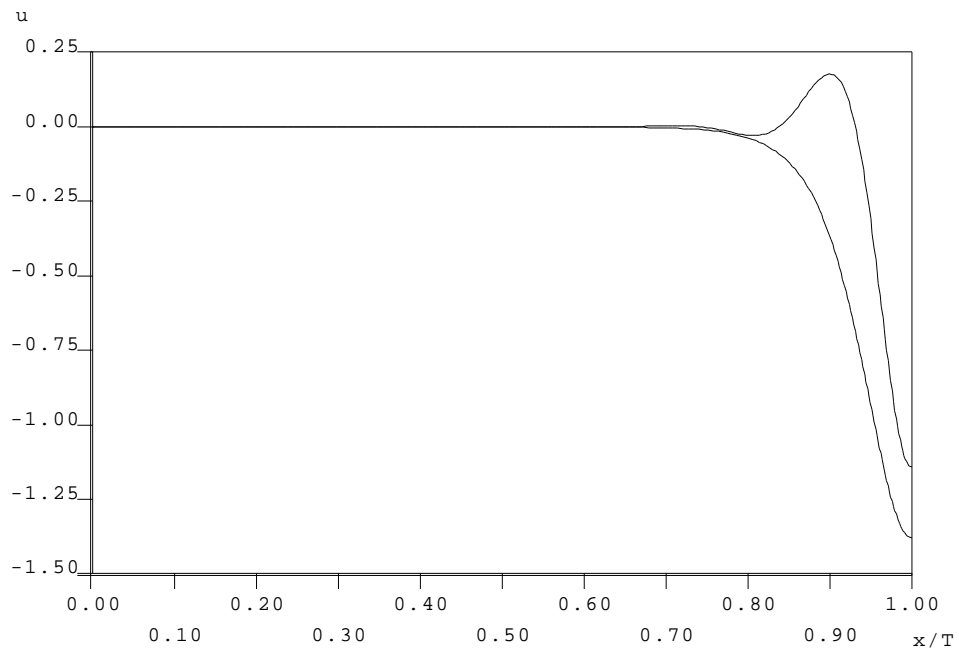


Figure 26.2: R_1 -reversible homoclinic orbits with oscillatory decay as $x \rightarrow -\infty$ (corresponding to label 6) and monotone decay (at label 10)

1	19	UZ	4	1.600000E+00	4.329404E-01	7.769491E-01
1	31	UZ	5	1.000000E+00	4.808488E-01	1.083298E+00
1	86	UZ	6	-9.664802E-10	5.158463E-01	1.258650E+00

contains the label of a limit point (ILP was set to 1 in c.rev.3, which corresponds to a “coalescence” of two reversible homoclinic orbits. The two solutions on either side of this limit point are displayed in Figure 26.3. The computation ends in a no-convergence point. The solution here is depicted in Figure 26.4. The lack of convergence is due to the large peak and trough of the solution rapidly moving to the left as $P \rightarrow -2$ (cf. Champneys & Spence (1993)).

Continuing from the initial solution in the other parameter direction

make fourth

we obtain the output

BR	PT	TY	LAB	PAR(1)	L2-NORM	MAX U(1)	...
1	7	UZ	8	1.600000E+00	3.701709E-01	3.836833E-01	
1	33	UZ	9	9.999980E-01	3.614405E-01	1.775035E-01	
1	93	UZ	10	-7.819855E-06	3.713007E-01	4.698309E-02	

which again ends at a no convergence error for similar reasons.

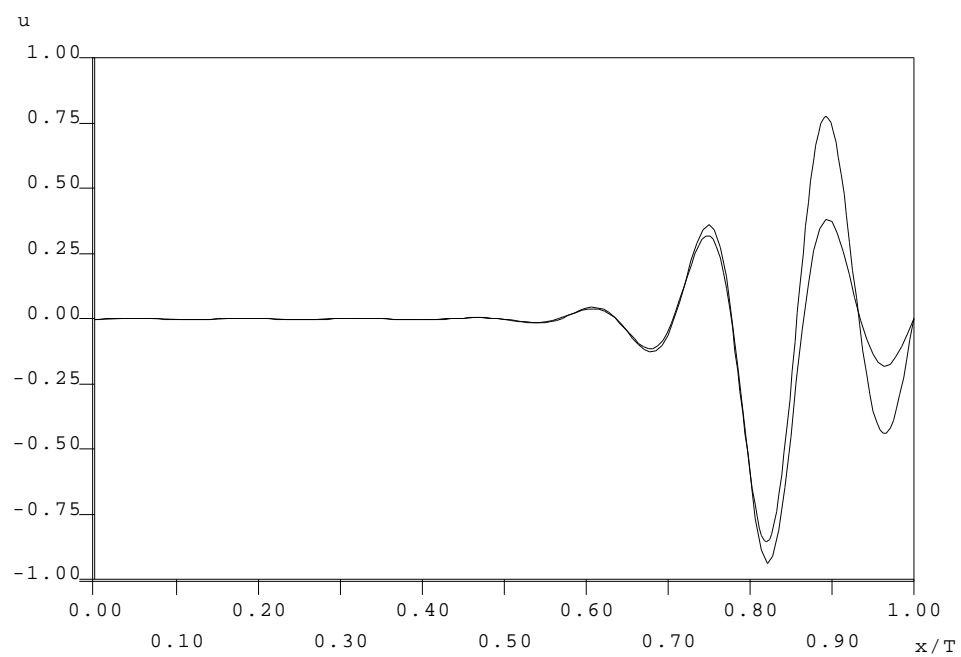


Figure 26.3: Two R_2 -reversible homoclinic orbits at $P = 1.6$ corresponding to labels 1 (smaller amplitude) and 5 (larger amplitude)

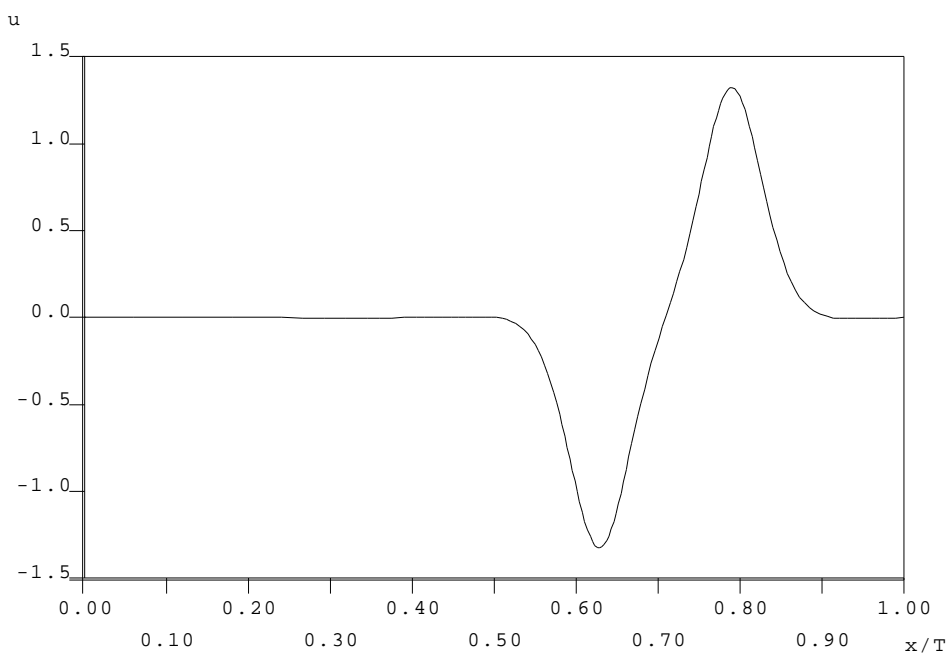


Figure 26.4: An R_2 -reversible homoclinic orbit at label 8

26.4 Detailed AUTO -Commands.

AUTO -COMMAND	ACTION
! mkdir rev cd rev demo('rev')	create an empty work directory change directory copy the demo files to the work directory
! cp rev.c.1 rev.c ! cp rev.dat.1 rev.dat us('rev') run(c='rev.1',h='rev.1',s='dat') sv('1')	get equations file to rev.c get the starting data to rev.dat use the starting data in rev.dat to create s.dat increase PAR(1) save output-files as b.1, s.1, d.1
run(c='rev.2',h='rev.2',s='1') ap('1')	continue in reverse direction; restart from s.1 append output-files to b.1, s.1, d.1
! cp rev.c.3 rev.c ! cp rev.dat.3 rev.dat us('rev') run(c='rev.3',h='rev.3',s='dat') sv('3')	get equations file with new value of PAR(11) get starting data with different reversibility use the starting data in rev.dat to create s.dat restart with different reversibility save output-files as b.3, s.3, d.3
run(c='rev.4',h='rev.4',s='3') ap('3')	continue in reverse direction; restart from s.3 append output-files to b.3, s.3, d.3

Table 26.1: Detailed AUTO -Commands for running demo rev.

Chapter 27

HomCont Demo : Homoclinic branch switching.

This demo illustrates homoclinic branch switching, which is an implementation of Lin's method (Lin 1990, Sandstede 1993, Yew 2001) as described in Oldeman et al. (2003). We use a direct branch switching method to switch from 1- to 2- and 3-homoclinic orbits near an inclination flip bifurcation in a model due to Sandstede, which was introduced in Chapter 21. This also shows how to obtain a homoclinic orbit through continuation of a periodic orbit born at a Hopf bifurcation. Thereafter, we illustrate homoclinic branch switching for the FitzHugh-Nagumo equations and a 5th-order Korteweg-De Vries model.

The equation files in these demos are written in C.

27.1 Branch switching at an inclination flip in Sandstede's model.

Consider the system (Sandstede 1995a)

$$\begin{aligned}\dot{x} &= ax + by - ax^2 - \alpha zx(2 - 3x), \\ \dot{y} &= bx + ay - \frac{3}{2}x(bx + ay) + \alpha z 2y, \\ \dot{z} &= cz + \mu x + 3xz + \alpha(x^2(1 - x) - y^2).\end{aligned}\tag{27.1}$$

as given in the file `sib.c`, where for simplicity we have set $\tilde{\mu} = 0$, $\beta = 1$ and $\gamma = 3$.

We study an inclination flip that exists for $a = 0.375$, $b = 0.625$ and $c = -0.75$. This corresponds to the situation where the eigenvalues of the equilibrium at the origin are $a + b = 1$, $a - b = -0.25$ and $c = -0.75$. Hence, the corresponding bifurcation diagram consists of a complicated structure involving a fan of infinitely many n -periodic and n -homoclinic orbits for arbitrary n and a region with horseshoe dynamics; see also Homburg & Krauskopf (2000) and the references therein.

This computation starts from an equilibrium at $(2/3, 0, 0)$, which exists for $a = \mu = \alpha = 0$. Also, b is set to 0.625 (the value we would like it to be) and c is set to -2.5 in `stpnt`. Choosing $c = -2$ at this stage leads to convergence problems. This equilibrium is not the one corresponding to the homoclinic orbit, but it is an equilibrium with complex eigenvalues, that

we can follow until it reaches a Hopf bifurcation. A periodic orbit emanates from this Hopf bifurcation and can be followed to the homoclinic orbit. However, first we need to change a from 0 to 0.375.

All the following commands, except for `demo('sib')` are contained within the file `'sib.auto'` which you can either execute in a batch mode by entering

```
> auto sib.auto
```

or step by step using

```
AUTO> demofile('sib.auto').
```

We start by copying the demo to the current work directory and running the first step

```
demo('sib')
ld('sib')
rn()
sv('1')
```

The equilibrium is followed in a until a (or `PAR(1)`) is at our desired value, 0.375.

BR	PT	TY	LAB	PAR(1)	...	U(1)	U(2)	U(3)
1	1	EP	1	0.00000E+00	...	6.66667E-01	0.00000E+00	0.00000E+00
1	5	EP	2	3.75000E-01	...	6.66667E-01	-1.33333E-01	0.00000E+00

The output is saved in the files `b.1`, `s.1` and `d.1`. Next we continue in α (`PAR(4)`) until a Hopf bifurcation is found:

```
rn(c='sib.2',s='1')
sv('2')
```

or, alternatively,

```
cc("IRS",2)
cc("ICP",[4])
rn(s='1')
sv('2')
```

BR	PT	TY	LAB	PAR(4)	...	U(1)	U(2)	U(3)
1	6	HB	3	3.18429E-01	...	6.54375E-01	-1.34754E-01	7.70102E-02

The output is saved in the files `b.2`, `s.2` and `d.2`. This Hopf bifurcation can then be continued into a periodic orbit. The periodic orbit eventually reaches a homoclinic bifurcation. We continue in $\mu=\text{PAR}(5)$ and `PAR(11)`, which corresponds to the period, and stop when the period is equal to 35.

```
rn(c='sib.3',s='2')
sv('3')
```

BR	PT	TY	LAB	PAR(5)	L2-NORM	...	PERIOD
3	10		5	-2.41881E-03	6.70569E-01	...	1.08975E+01
						...	
3	40		8	-1.29495E-02	6.14547E-01	...	1.41297E+01
						...	
3	81	EP	13	-1.04657E-04	4.01829E-01	...	3.50000E+01

The output is saved in the files `b.3`, `s.3` and `d.3`. Note that μ first decreases and then increases towards 0, which is precisely what we expect in this model, as homoclinic orbits occur on the line $\mu = 0$ in the (α, μ) -plane. It is now instructive to look at a phase space diagram to see what is going on.

`plot('3')`

Selecting 'solution' for Type, [5,6,7,8,9,10,11,12,13] for Label, [0] for X and [1] for Y, we obtain the diagram depicted in Figure 27.1(a), where the periodic orbit grows from the Hopf equilibrium to a homoclinic orbit.

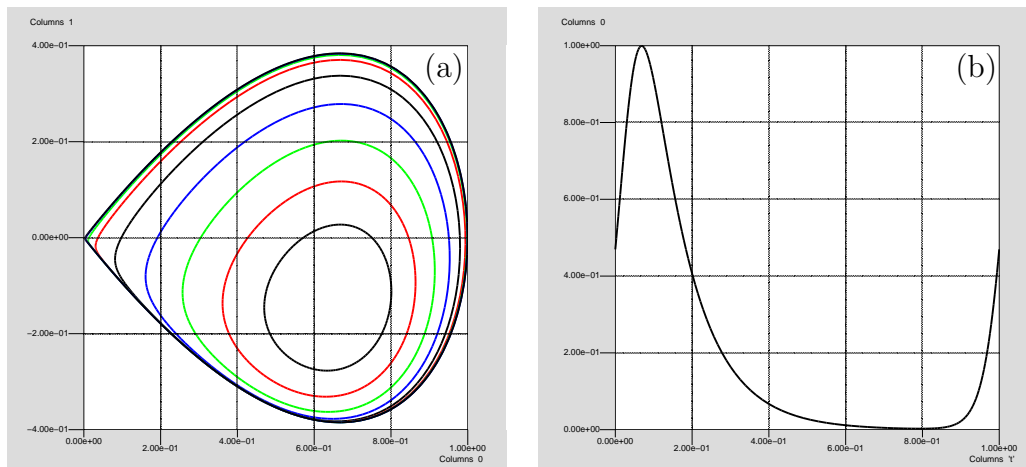


Figure 27.1: Periodic orbit growing from a Hopf bifurcation to a homoclinic orbit (a). The unshifted homoclinic orbit (b).

Note however, that the homoclinic orbit has the wrong left-hand and right-hand end points. This can be seen by plotting the solution corresponding to Label [13] using 't' vs. 'x' (coordinate [0]), as depicted in Figure 27.1(b).

Hence, in order to continue this as a real homoclinic we have to give HomCont special instructions, to do a phase-shift in time. This can be done by setting `ISTART=4`. Moreover, since we have not specified the value of the equilibrium at the origin in `sib.c`, we need to set `IEQUIB=1` to let HomCont detect the equilibrium. Note that in this case this is not strictly necessary; however, we do this for instructional purposes.

Now we use HomCont to continue the homoclinic orbit in c and μ (`PAR(3)`, `PAR(5)`), to get the desired value $c = -2.0$.

```
rn(c='sib.4',h='sib.shift',s='3')
sv('4')
```

BR	PT	TY	LAB	PAR(3)	L2-NORM	...	PAR(5)
3	51	EP	14	-2.00000E+00	4.01890E-01	...	2.66146E-09

The output is saved in the files `b.4`, `s.4` and `d.4`. Note that `PAR(5)= μ` remains zero, which is exactly what we expect.

Next we want to add a solution to the adjoint equation to this solution. This is achieved by making the change `ITWIST = 1` saved in `h.sib.twist`. Also, we set `ISTART` to 1 to tell HomCont that it should not try to shift the orbit anymore.

```
rn(c='sib.5',h='sib.twist',s='4')
sv('5')
```

or, alternatively,

```
cc("IRS",14)
cc("ICP",[5,8])
cc("NMX",2)
hch("ITWIST",1)
hch("ISTART",1)
rn(s='4')
sv('5')
```

where `hch` means “change HomCont constant”. The output is stored in `b.5`, `s.5` and `d.5`.

BR	PT	TY	LAB	PAR(5)	L2-NORM	...	PAR(8)
3	2	EP	15	2.66146E-09	4.01890E-01	...	1.00000E-02

Here `PAR(8)` is a dummy (unused) parameter and μ just stays where it is. Now that we have obtained the solution of the adjoint equation, we are able to detect inclination flips. This can be achieved by setting `NPSI` to 1, `IPSI(1)` to 13, and monitoring `PAR(33)`.

```
rn(c='sib.6',h='sib.if',s='5')
sv('6')
```

BR	PT	TY	LAB	PAR(4)	L2-NORM	...	PAR(5)	PAR(33)
3	19	UZ	16	7.11774E-02	4.01890E-01	...	1.24376E-11	-2.36702E-07

The output is stored in `b.6`, `s.6` and `d.6`. Hence an inclination flip was found at $\alpha = 0.711774$.

Now we are ready to perform homoclinic branch switching, using the techniques described in (Oldeman et al. 2003). Our first aim is to find a 2-homoclinic orbit. The ingredients we need are: a homoclinic orbit where n -homoclinic orbits are close by, and the solution to the adjoint equation to obtain the Lin vector. Since both ingredients are there, we can now continue in μ , ε_1 and T_1 , to obtain the initial Lin gap. Recall from Chapter 20 that the Lin gaps ε_i correspond to `PAR(20+i*2)` and the time intervals T_i correspond to `PAR(21+i*2)`. We stop when $\varepsilon_1 = 0.2$. We need to specify `ITWIST=2`, to tell HomCont we aim to find a 2-homoclinic orbit, so that it will split it up in three parts with two potential Lin gaps. We effectively have a 9-dimensional system at this point.

```
rn(c='sib.7',h='sib.hbs2',s='6')
sv('7')
```

BR	PT	TY	LAB	PAR(21)	L2-NORM	...	PAR(22)	PAR(5)
3	10		18	3.45897E+01	4.46818E-01	...	7.87712E-07	-1.55885E-11
3	20		19	2.73699E+01	4.46818E-01	...	2.91119E-05	-1.63974E-09
3	30		20	1.73720E+01	4.46817E-01	...	4.42273E-03	-3.10167E-05
3	38	EP	21	1.01451E+01	4.46796E-01	...	2.00000E-01	-1.48615E-02

The output is stored in `b.7`, `s.7` and `d.7`. Here we see that T_1 , the time it takes to make the first loop with respect to the Poincaré section, decreases. This is illustrated in Figure 27.2. Next we are ready to close this gap, by continuing in α , μ , and ε_1 , while keeping T_1 at a constant value.

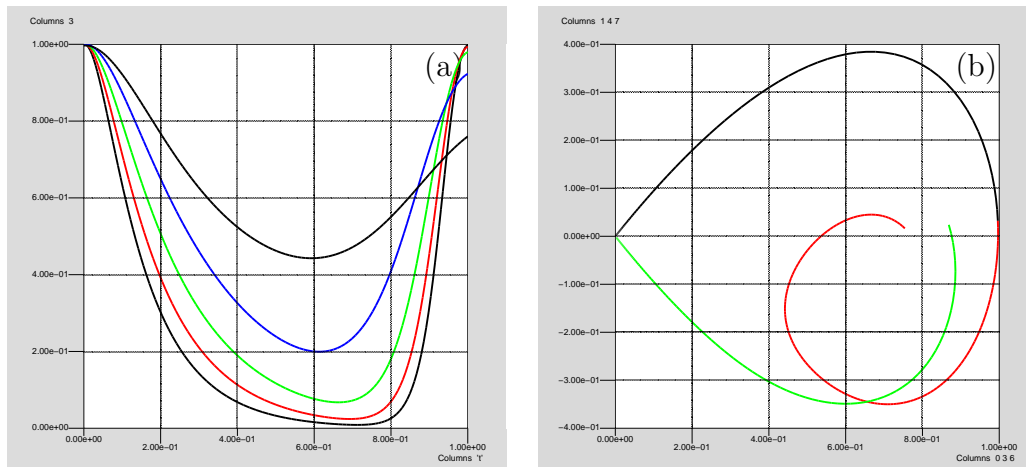


Figure 27.2: Behaviour of the second piece of the ‘broken homoclinic orbit’ when creating a Lin gap (a). Projection of the “broken homoclinic orbit” onto the (x, y) -plane, where $\varepsilon_1 = 0.2$. To include all the pieces necessary to obtain this figure, the “X” box must contain $[0, 3, 6]$ and the “Y” box must contain $[1, 4, 7]$ (b).

```
rn(c='sib.8',h='sib.hbs2',s='7')
ap('6')
```

BR	PT	TY	LAB	PAR(4)	L2-NORM	...	PAR(5)	PAR(22)
3	3	UZ	22	7.40000E-02	4.46781E-01	...	-1.43162E-02	1.93746E-01
3	32	EP	23	1.98414E-01	4.46590E-01	...	-6.05495E-03	2.29300E-06

The output is appended to `b.6`, `s.6` and `d.6`. Now we have obtained a 2-homoclinic orbit at label 24. However, the homoclinic orbit is still split in three parts. We can switch back to a normal orbit by setting `ITWIST` back to 0 and continuing in the usual way. Here we continue back to the inclination flip point in α and μ .

```
rn(c='sib.9',h='sib.hom',s='6')
ap('6')
```


BR	PT	TY	LAB	PAR(4)	L2-NORM	...	PAR(5)
3	7	UZ	24	1.50000E-01	4.94490E-01	...	-3.60248E-03
3	30	EP	25	7.61403E-02	4.98746E-01	...	-2.64847E-06

So the 2-homoclinic orbit converges back to the 1-homoclinic orbit at the inclination flip bifurcation. The output is appended to **b.6**, **s.6** and **d.6**. The resulting 2-homoclinic orbits can be seen using

`plot('6')`

and is depicted in Figure 27.3(a).

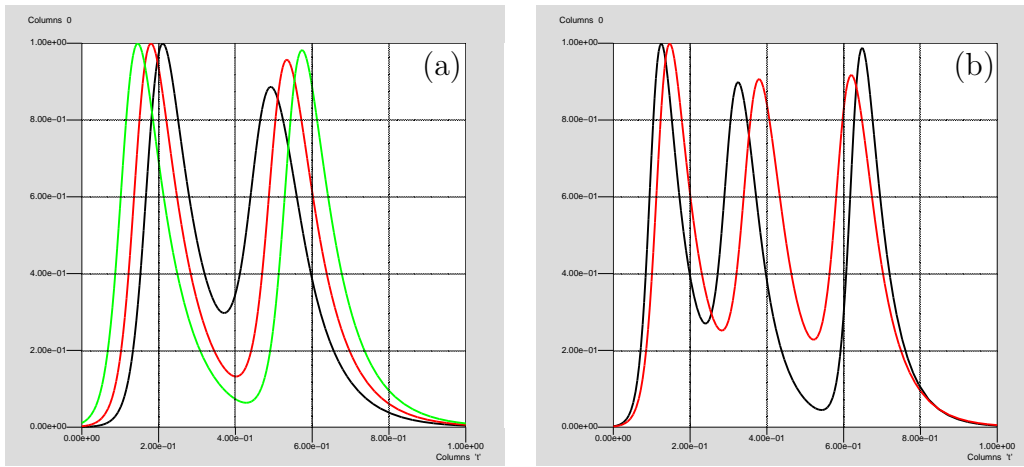


Figure 27.3: The 2-homoclinic orbit as a is changed (a). The two different 3-homoclinic orbits (b).

Next, we aim to find a 3-homoclinic orbit. To do so, we restart at the inclination flip point at label 16 and set `ITWIST=3`. Moreover, we need to continue in one more gap, $\varepsilon_2 = \text{PAR}(24)$ and, once again, stop when $\varepsilon_1 = \text{PAR}(22) = 0$. Note that the dimension of the boundary value problem we continue is now equal to 12. This is not to be confused with the setting of `NDIM=3` in the parameter file, because `HomCont` handles this internally.

`rn(c='sib.10',h='sib.hbs3',s='6')`
`sv('10')`

BR	PT	TY	LAB	PAR(21)	...	PAR(22)	PAR(24)	PAR(5)
3	10		26	3.45896E+01	...	7.87894E-07	6.42157E-07	-1.06346E-11
3	20		27	2.73699E+01	...	2.91126E-05	6.51591E-07	-1.63655E-09
3	30		28	1.73719E+01	...	4.42289E-03	1.44090E-04	-3.10188E-05
3	38	EP	29	1.01451E+01	...	2.00000E-01	6.97445E-02	-1.48615E-02

The output is stored in **b.10**, **s.10** and **d.10**. Now we need to subsequently close the Lin gaps. Our strategy is to keep T_1 fixed. We first continue in α , μ , ε_1 and ε_2 until $\varepsilon_1 = 0$.

```
rn(c='sib.11',h='sib.hbs3',s='10')
ap('6')
```

BR	PT	TY	LAB	PAR(4)	...	PAR(5)	PAR(22)	PAR(24)
3	6	UZ	30	8.20000E-02	...	-1.29790E-02	1.76995E-01	6.37184E-02
3	32	EP	31	1.98414E-01	...	-6.05495E-03	2.30717E-06	3.62449E-02

The output is appended to `b.6`, `s.6` and `d.6`. Note that this continuation is very similar to the one where we found a 2-homoclinic orbit. In fact we have now found a 2-homoclinic orbit (numerically) followed by a ‘broken’ 1-homoclinic orbit; only the mesh is not aligned.

The next step is to close the gap corresponding to ε_2 to obtain a 3-homoclinic orbit. We replace the continuation parameter ε_1 by T_2 , because T_2 (PAR(23)) still has to be decreased from its high value (35) and ε_1 needs to stay at 0.

```
rn(c='sib.12',h='sib.hbs3',s='6')
ap('6')
```

BR	PT	TY	LAB	PAR(4)	...	PAR(5)	PAR(23)	PAR(24)
3	16	UZ	32	1.98395E-01	...	-6.05536E-03	2.01311E+01	1.82491E-08
3	24	UZ	33	1.80000E-01	...	-6.50293E-03	1.27554E+01	-3.14294E-02
3	30	UZ	34	1.66990E-01	...	-6.89269E-03	9.41745E+00	-1.03179E-06
3	32	EP	35	1.78172E-01	...	-6.55364E-03	9.50300E+00	-7.20367E-02

The output is appended to `b.6`, `s.6` and `d.6`. Note that we have found two zeros of PAR(24), at labels 32 and 34, respectively. The two zeros correspond to two different 3-homoclinic orbits, which, when followed from periodic orbits, both emanate from from the same saddle-node bifurcation. These two 3-homoclinic orbits are depicted in Figure 27.3(b). We can follow both of these back to the inclination flip point, by setting ITWIST back to 0:

```
rn(c='sib.13',h='sib.hom',s='6')
ap('6')
```

BR	PT	TY	LAB	PAR(4)	L2-NORM	...	PAR(5)
3	13	UZ	36	1.29999E-01	5.04807E-01	...	-2.33902E-03
3	30	EP	37	9.27258E-02	5.06560E-01	...	-2.76788E-04

```
rn(c='sib.14',h='sib.hom',s='6')
ap('6')
```

BR	PT	TY	LAB	PAR(4)	L2-NORM	...	PAR(5)
3	7	UZ	38	1.45000E-01	5.47347E-01	...	-4.79400E-03
3	30	EP	39	8.39401E-02	5.52605E-01	...	-7.36741E-05

All the output is appended to `b.6`, `s.6` and `d.6`. The bifurcation diagram and the paths we followed when closing the Lin gaps are depicted in Figure 27.4. It is possible and straightforward to obtain 4, 5, 6, ...-homoclinic orbits by extending the above strategy.

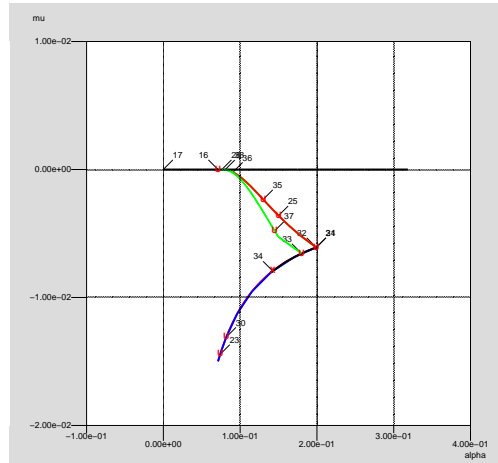


Figure 27.4: Parameter space diagram near an inclination flip. The curve through label 17 corresponds to a 1-homoclinic orbit. The opening of the Lin gaps occurs along the vertical line from label 16 to label 23. The curves through labels 23 and 30 denote the path that is followed when closing the Lin gaps. The (approximately overlaid) curves through labels 25 and 35 correspond to the 2- and one of the 3-homoclinic orbits. Finally, the curve through label 37 corresponds to the other 3-homoclinic orbit, which was obtained for $\text{PAR}(22)=T_2 = 12.03201$.

27.2 Branch switching for a Shil'nikov type homoclinic orbit in the FitzHugh-Nagumo equations.

The FitzHugh-Nagumo (FHN) equations (FitzHugh 1961, Nagumo, Arimoto & Yoshizawa 1962) are a simplified version of the Hodgkin-Huxley equations (Hodgkin & Huxley 1952). They model nerve axon dynamics and are given by

$$\begin{aligned} u_t &= u_{xx} - f_a(u) - w, \\ w_t &= \epsilon(u - \gamma w), \end{aligned} \tag{27.2}$$

where

$$f_a(u) = u(u - a)(u - 1).$$

Travelling wave solutions of the form $(u, w)(x, t) = (u, w)(\xi)$, where $\xi = x + ct$ are solutions of the following ODE system:

$$\begin{aligned} \dot{u} &= v, \\ \dot{v} &= cv + f_a(u) + w, \\ \dot{w} &= \frac{\epsilon}{c}(u - \gamma w). \end{aligned} \tag{27.3}$$

In particular we consider solitary wave solutions of (27.2). These correspond to orbits homoclinic to $(u, v, w) = 0$ in system (27.3). In our numerical example we keep $\gamma = 0$.

We aim to find a 2-homoclinic orbit at a Shil'nikov bifurcation. All the commands given here are in the file `fnb.auto`. First we obtain a homoclinic orbit using a homotopy technique (see Friedman, Doedel & Monteiro (1994)), using `ISTART=3`, for the parameter values $c = 0.21$, $a = 0.2$, $\epsilon = 0.0025$.

```
demo('sib')
ld('fnb')
rn()
sv('1')
```

Among the output we see:

BR	PT	TY	LAB	PERIOD	L2-NORM	...	PAR(17)
1	20	UZ	3	2.92257E+01	2.37916E-01	...	-1.68000E-09

and a zero of `PAR(17)` means that a zero of an artificial parameter has been located and the right-hand end point of the corresponding solution belongs to the plane that is tangent to the stable manifold at the saddle. This point still needs to come closer to the equilibrium, which we can achieve by further increasing the period to 300, while keeping `PAR(17)` at 0:

```
rn(c='fnb.2',h='fnb.1',s='1')
sv('2')
```

BR	PT	TY	LAB	PERIOD	L2-NORM	...	PAR(2)
1	190	UZ	10	3.00000E+02	7.37932E-02	...	1.79286E-01

Next we stop using the homotopy technique and increase the period even further, to 1000.

```
rn(c='fnb.3',h='fnb.3',s='2')
sv('3')
```

BR	PT	TY	LAB	PERIOD	L2-NORM	...	PAR(2)
1	80	UZ	13	1.00000E+03	4.04183E-02	...	1.79286E-01

A continuation in `PAR(2)=a` and `PAR(1)=c` needs to be performed to arrive at the place where we wish to find a 2-homoclinic orbit: $a = 0$. At the same time we monitor `PAR(22)` to locate Belyakov points.

```
rn(c='fnb.4',h='fnb.4',s='3')
sv('4')
```

BR	PT	TY	LAB	PAR(2)	L2-NORM	...	PAR(1)	PAR(22)
1	6	UZ	15	1.31812E-01	3.28710E-02	...	2.17166E-01	-6.31243E-06
1	23	UZ	19	-8.54548E-08	1.56158E-02	...	2.74218E-01	-9.88772E-02

Hence, there exists a Belyakov point at $(a, c) = (0.131812, 0.21766)$. At label 19 we have a lower value of a than at the Belyakov point, and by inspection of the file `d.4` we can observe that the equilibrium has one positive eigenvalue and a complex conjugate pair of eigenvalues with negative real part, and conclude that this orbit is of Shil'nikov type. Before starting the homoclinic branch switching, we calculate the adjoint to obtain a 'Lin vector':

```
rn(c='fmb.5',h='fmb.5',s='4')
sv('5')
```

BR	PT	TY	LAB	PAR(9)	L2-NORM	...	PAR(3)
1	2	EP	28	-1.00000E+00	1.56158E-02	...	2.50000E-03

Next, we continue in the time T_1 (PAR(21)), the gap ε_1 (PAR(22)) and c (PAR(1)), and by setting ISTART=-2 we try to locate a 2-homoclinic orbit:

```
rn(c='fmb.6',h='fmb.6',s='5')
sv('6')
```

In fact we find many of them, exactly as is predicted by the theory:

BR	PT	TY	LAB	PAR(21)	...	PAR(1)	PAR(22)
...							
1	175	UZ	46	1.64818E+02	...	2.74218E-01	4.59218E-16
1	180	UZ	47	1.44759E+02	...	2.74218E-01	-1.43728E-14
1	184	UZ	48	1.24939E+02	...	2.74218E-01	1.55506E-13
1	189	UZ	49	1.04615E+02	...	2.74218E-01	-2.37665E-11
1	193	UZ	50	8.53538E+01	...	2.74218E-01	1.02165E-11
1	198	UZ	51	6.37899E+01	...	2.74218E-01	-5.74204E-14

Each of these homoclinic orbits differ by about 20 in the value T_1 . This is about the time it takes to make one half-turn close to and around the equilibrium, so that orbits differ by the number of half turns around the equilibrium before a big excursion in phase space. Note that the variation of c is so small that it does not appear.

A plot of T_1 vs. ε_1 gives insight into how the gap is opened and closed in the continuation process. This is depicted in Figure 27.5. We are now in a position to continue each of these

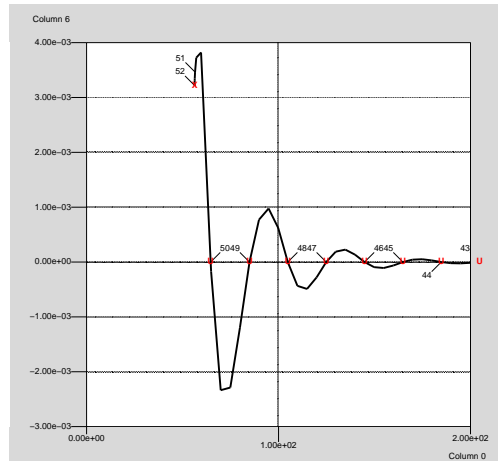


Figure 27.5: A plot of ε_1 as a function of T_1 during our computation of Shil'nikov-type two-homoclinic orbits. Each zero corresponds to a different orbit.

orbits as a normal homoclinic orbit by setting ISTART=1 and ITWIST=0. We leave this as an exercise to the reader.

27.3 Branch switching to a 3-homoclinic orbit in a 5th-order Korteweg-De Vries model

In Champneys & Groves (1997) the following water wave model was considered:

$$\frac{2}{15}r'''' - br'' + ar + \frac{3}{2}r^2 - \frac{1}{2}(r')^2 + [rr']' = 0. \quad (27.4)$$

It represents solitary-wave solutions $r(x + at)$, $r \rightarrow 0$ as $x \rightarrow \pm\infty$ of the 5th-order PDE

$$r_t + \frac{2}{15}r_{xxxx} - br_{xxx} + 3rr_x + 2r_xr_{xx} + rr_{xxx} = 0,$$

where a is the wave speed. The ODE corresponds to a Hamiltonian system with Hamiltonian

$$H = -\frac{1}{2}q_1^3 - \frac{1}{2}aq_1^2 + p_1q_2 - \frac{1}{2}bq_2^2 + \frac{15}{4}p_2^2 + \frac{1}{2}q_2^2q_1$$

and

$$q_1 = r, \quad q_2 = r', \quad p_1 = -\frac{2}{15}r''' + br' - rr', \quad p_2 = \frac{2}{15}r''.$$

System (27.4) is also reversible under the transformation

$$t \mapsto -t, (q_1, q_2, p_1, p_2) \mapsto (q_1, -q_2, -p_1, p_2),$$

but we do not exploit the reversible structure (IREV=0), and instead use it as an example of Hamiltonian system. This system exhibits an orbit flip for a reversible Hamiltonian system. In Hamiltonian systems, homoclinic orbits are codimension-zero phenomena, and we have to add an additional parameter λ that breaks the Hamiltonian structure in this system, by introducing artificial friction. Thus, the actual system of equations that is used for continuation is

$$\dot{x} = (\lambda I + J)\nabla H(x),$$

where $x = (q_1, q_2, p_1, p_2)$ and J is the usual skew symmetric matrix in \mathbb{R}^4 . It is now possible to continue a homoclinic orbit in HomCont in two parameters (λ and either a or b); see also Beyn (1990).

An explicit solution exists for $a = 3/5(2b + 1)(b - 2)$, $b \geq -1/2$, and it is given by

$$r(t) = 3(b + \frac{1}{2})\text{sech}^2\left(\left[\frac{3}{4}(2b + 1)\right]^{1/2}t\right).$$

It corresponds to a reversible orbit flip for $b > 2$ ($a > 0$). We start from this explicit solution, using ISTART=2, for $a = 3$ and $b = (\sqrt{65} + 3)/4$:

```
demo('kdv')
ld('kdv')
rn()
sv('1')
```

BR	PT	TY	LAB	PAR(1)	L2-NORM	...	PAR(3)
1	1	EP	1	3.00000E+00	5.56544E+00	...	0.00000E+00
1	2	EP	2	3.04959E+00	5.49141E+00	...	1.77919E-17

Here $\text{PAR}(1)=a$, $\text{PAR}(2)=b$, and $\text{PAR}(3)=\lambda$. We have only done a very small continuation to give AUTO a chance to create a good mesh and avoid convergence problems later. Next, we set $\text{ITWIST}=1$ and calculate the adjoint:

rn(c='kdv.2',h='kdv.2',s='1')
sv('2')

BR	PT	TY	LAB	PAR(2)	L2-NORM	...	PAR(9)
1	2	EP	3	2.76556E+00	5.49140E+00	...	-7.62944E-08

We now need to move back to the orbit flip at $a = 3$:

rn(c='kdv.3',h='kdv.3',s='2')
sv('3')

BR	PT	TY	LAB	PAR(1)	L2-NORM	...	PAR(3)
1	14	UZ	5	3.00000E+00	5.47612E+00	...	1.47274E-09

Now all preparations are done to start homoclinic branch switching. This is very similar to the technique used in Sandstede's model in Section 27.1; to find a 3-homoclinic orbit, we open 2 Lin gaps, until $T_1 = 3.5$, while also varying $\lambda=\text{PAR}(3)$.

rn(c='kdv.4',h='kdv.4',s='3')
sv('4')

BR	PT	TY	LAB	PAR(3)	...	PAR(21)	PAR(22)	PAR(24)
1	13		8	6.31458E-10	...	1.65469E+01	-8.57681E-08	-7.30773E-07
1	23	UZ	9	1.46493E-09	...	9.92489E+00	-5.84373E-12	1.93098E-07
1	26		10	4.01320E-09	...	6.92406E+00	2.59555E-07	7.47534E-07
1	33	EP	11	2.15487E-06	...	3.50000E+00	7.92587E-04	3.98390E-04

We then look for an orbit with $a < 3$ and close the gap corresponding to $\varepsilon_1=\text{PAR}(22)$, for decreasing a .

rn(c='kdv.5',h='kdv.5',s='4')
sv('5')

BR	PT	TY	LAB	PAR(2)	...	PAR(3)	PAR(22)	PAR(24)
1	10		12	2.58030E+00	...	2.15869E-06	7.65037E-04	3.82464E-04
1	13	UZ	13	2.32044E+00	...	4.02730E-11	1.17522E-10	1.69655E-08
1	20	EP	14	-1.14985E-01	...	-8.87194E-04	-7.18231E-01	-3.31153E-01

and finally close the gap corresponding to $\varepsilon_2=\text{PAR}(24)$,

```
rn(c='kdv.6',h='kdv.6',s='5')
sv('6')
```

BR	PT	TY	LAB	PAR(2)	...	PAR(3)	PAR(23)	PAR(24)
1	35		15	2.31893E+00	...	-2.16070E-08	7.69046E+00	-1.08126E-05
1	51	UZ	16	2.34039E+00	...	2.83533E-07	3.47976E+00	1.42651E-04
1	58	UZ	17	3.08085E+00	...	1.84952E-12	3.50004E+00	-1.64827E-10
1	70	EP	18	3.08870E+00	...	-8.10422E-08	5.87541E+00	-4.82991E-05

so that a three-homoclinic orbit is found. Here the zero at label 16 is the one we are looking for. At label 17, $a=\text{PAR}(1)$ has changed considerably to the extent that $a > 3$ and a second 3-homoclinic orbit is found. Note that for all zeros of $\text{PAR}(24)=\varepsilon_2$, the parameter $\lambda=\text{PAR}(3)$ is also zero (within AUTO accuracy), which it has to be to remain within the original Hamiltonian system. Setting $\text{ISTART}=1$, a normal “trivial” continuation (with $\text{NMX}=1$) of the orbit corresponding to label 16 lets HomCont produce a proper concatenated 3-homoclinic orbit:

```
rn(c='kdv.7',h='kdv.7',s='6')
sv('7')
```

BR	PT	TY	LAB	PAR(2)	L2-NORM	...	PAR(3)
1	1	EP	19	2.34039E+00	7.51157E+00	...	2.83533E-07

This 3-homoclinic orbit is depicted in Figure 27.6.

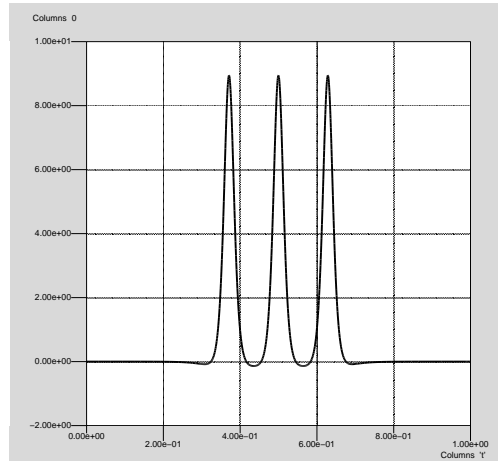


Figure 27.6: A 3-homoclinic orbit in a 5th-order Hamiltonian Korteweg-De Vries model.

Bibliography

- Alexander, J. C., Doedel, E. J. & Othmer, H. G. (1990), ‘On the resonance structure in a forced excitable system’, *SIAM J. Appl. Math.* **50**, No. 5, 1373–1418.
- Aronson, D. G. (1980), Density dependent reaction-diffusion systems, in ‘Dynamics and Modelling of Reactive Systems’, Academic Press, pp. 161–176.
- Bai, F. & Champneys, A. (1996), ‘Numerical detection and continuation of saddle-node homoclinic orbits of codimension one and codimension two’, *J. Dyn. Stab. Sys.* **11**, 327–348.
- Beyn, W.-J. (1990), ‘The numerical computation of connecting orbits in dynamical systems’, *IMA J. Num. Anal.* **9**, 379–405.
- Beyn, W.-J. & Doedel, E. J. (1981), ‘Stability and multiplicity of solutions to discretizations of nonlinear ordinary differential equations’, *SIAM J. Sci. Stat. Comput.* **2**(1), 107–120.
- Buryak, A. & Akhmediev, N. (1995), ‘Stability criterion for stationary bound states of solitons with radiationless oscillating tails’, *Physical Review E* **51**, 3572–3578.
- Champneys, A. & Kuznetsov, Y. (1994), ‘Numerical detection and continuation of codimension-two homoclinic bifurcations’, *Int. J. Bifurcation & Chaos* **4**, 795–822.
- Champneys, A. & Spence, A. (1993), ‘Hunting for homoclinic orbits in reversible systems: a shooting technique’, *Adv. Comp. Math.* **1**, 81–108.
- Champneys, A., Kuznetsov, Y. & Sandstede, B. (1996), ‘A numerical toolbox for homoclinic bifurcation analysis’.
- Champneys, A. R. & Groves, M. D. (1997), ‘A global investigation of a solitary wave solutions to a fifth-order two-parameter model equation for water waves.’, *J. Fluid Mechanics* **342**, 199–229.
- de Boor, C. & Swartz, B. (1973), ‘Collocation at gaussian points’, *SIAM J. Numer. Anal.* **10**, 582–606.
- Doedel, E. J. (1981), ‘AUTO, a program for the automatic bifurcation analysis of autonomous systems’, *Cong. Numer.* **30**, 265–384.
- Doedel, E. J. (1984), ‘The computer-aided bifurcation analysis of predator-prey models’, *J. Math. Biol.* **20**, 1–14.

- Doedel, E. J. & Heinemann, R. F. (1983), ‘Numerical computation of periodic solution branches and oscillatory dynamics of the stirred tank reactor with $a \rightarrow b \rightarrow c$ reactions’, *Chem. Eng. Sci.* **38**, No. 9, 1493–1499.
- Doedel, E. J. & Kernévez, J. P. (1986a), AUTO: Software for continuation problems in ordinary differential equations with applications, Technical report, California Institute of Technology. Applied Mathematics.
- Doedel, E. J. & Kernévez, J. P. (1986b), A numerical analysis of wave phenomena in a reaction diffusion model, in H. G. Othmer, ed., ‘Nonlinear Oscillations in Biology and Chemistry’, Vol. 66, Springer Verlag, pp. 261–273.
- Doedel, E. J. & Wang, X. J. (1995), AUTO94 : Software for continuation and bifurcation problems in ordinary differential equations, Technical report, Center for Research on Parallel Computing, California Institute of Technology, Pasadena CA 91125. CRPC-95-2.
- Doedel, E. J., Aronson, D. G. & Othmer, H. G. (1991), ‘The dynamics of coupled current-biased Josephson junctions II’, *Int. J. Bifurcation and Chaos* **1**, No. 1, 51–66.
- Doedel, E. J., Champneys, A. R., Fairgrieve, T. F., Kuznetsov, Y. A., Sandstede, B. & Wang, X. J. (1997), AUTO97 : Software for continuation and bifurcation problems in ordinary differential equations, Technical report, California Institute of Technology, Pasadena CA 91125.
- Doedel, E. J., Friedman, M. & Monteiro, A. (1993), On locating homoclinic and heteroclinic orbits, Technical report, Cornell Theory Center; Center for Applied Mathematics, Cornell University.
- Doedel, E. J., Keller, H. B. & Kernévez, J. P. (1991a), ‘Numerical analysis and control of bifurcation problems: (I) Bifurcation in finite dimensions’, *Int. J. Bifurcation and Chaos* **1**(3), 493–520.
- Doedel, E. J., Keller, H. B. & Kernévez, J. P. (1991b), ‘Numerical analysis and control of bifurcation problems: (II) Bifurcation in infinite dimensions’, *Int. J. Bifurcation and Chaos* **1**(4), 745–772.
- Doedel, E. J., Paffenroth, R. C., Champneys, A. R., Fairgrieve, T. F., Kuznetsov, Y. A., Oldeman, B. E., Sandstede, B. & Wang, X. J. (2000), AUTO2000 : Software for continuation and bifurcation problems in ordinary differential equations, Technical report, California Institute of Technology, Pasadena CA 91125.
- Fairgrieve, T. F. (1994), The computation and use of Floquet multipliers for bifurcation analysis, PhD thesis, University of Toronto.
- Fairgrieve, T. F. & Jepson, A. D. (1991), ‘O.K. Floquet multipliers’, *SIAM J. Numer. Anal.* **28**, No. 5, 1446–1462.
- FitzHugh, R. (1961), ‘Impulses and physiological states in theoretical models of nerve membrane’, *Biophys. J.* **1**, 445–446.

- Freire, E., Rodríguez-Luis, A., Gamero, E. & Ponce, E. (1993), ‘A case study for homoclinic chaos in an autonomous electronic circuit: A trip from Takens–Bogdanov to Hopf–Shilnikov’, *Physica D* **62**, 230–253.
- Friedman, M., Doedel, E. J. & Monteiro, A. C. (1994), ‘On locating connecting orbits’, *Applied Math. And Comp.* **65**(1–3), 231–239.
- Friedman, M. J. & Doedel, E. J. (1991), ‘Numerical computation and continuation of invariant manifolds connecting fixed points’, *SIAM J. Numer. Anal.* **28**, 789–808.
- Henderson, M. E. & Keller, H. B. (1990), ‘Complex bifurcation from real paths’, *SIAM J. Appl. Math.* **50**, No. 2, 460–482.
- Hodgkin, A. L. & Huxley, A. F. (1952), ‘A quantitative description of membrane current and its applications to conduction and excitation in nerve’, *J. Physiol.* **117**, 500–544.
- Holodniok, M., Kneřlik, P. & Kubíček, M. (1987), Continuation of periodic solutions in parabolic differential equations, in T. Küpper, R. Seydel & H. Troger, eds, ‘Bifurcation: Analysis, Algorithms, Applications’, Vol. INSM 79, Birkhäuser, Basel, pp. 122–130.
- Homburg, A. & Krauskopf, B. (2000), ‘Resonant homoclinic flip bifurcations’, *J. Dyn. Diff. Eqns.* **12**(4), 807–850.
- Hunt, G. W., Bolt, H. M. & Thompson, J. M. T. (1989), ‘Structural localization phenomena and the dynamical phase-space analogy’, *Proc. Roy. Soc. Lond. A* **425**, 245–267.
- Keller, H. B. (1977), Numerical solution of bifurcation and nonlinear eigenvalue problems, in P. H. Rabinowitz, ed., ‘Applications of Bifurcation Theory’, Academic Press, pp. 359–384.
- Keller, H. B. (1986), *Lectures on Numerical Methods in Bifurcation Problems*, Springer Verlag. Notes by A. K. Nandakumaran and Mythily Ramaswamy, Indian Institute of Science, Bangalore.
- Kernévez, J. P. (1980), *Enzyme Mathematics*, North-Holland Press, Amsterdam.
- Khibnik, A. I., Roose, D. & Chua, L. O. (1993), ‘On periodic orbits and homoclinic bifurcations in Chua’s circuit with a smooth nonlinearity’, *Int. J. Bifurcation and Chaos* **3**, No. 2, 363–384.
- Khibnik, A., Kuznetsov, Y., Levitin, V. & Nikolaev, E. (1993), ‘Continuation techniques and interactive software for bifurcation analysis of ODEs and iterated maps’, *Physica D* **62**, 360–371.
- Koper, M. (1994), Far-from-equilibrium phenomena in electrochemical systems, PhD thesis, Universiteit Utrecht, The Netherlands.
- Koper, M. (1995), ‘Bifurcations of mixed-mode oscillations in a three-variable autonomous Van der Pol–Duffing model with a cross-shaped phase diagram’, *Physica D* **80**, 72–94.

- Lentini, M. & Keller, H. B. (1980), ‘The Von Karman swirling flows’, *SIAM J. Appl. Math.* **38**, 52–64.
- Lin, X.-B. (1990), ‘Using Melnikov’s method to solve Silnikov’s problems’, *Proc. Royal Soc. Edinburgh* **116A**, 295–325.
- Lorenz, J. (1982), Nonlinear boundary value problems with turning points and properties of difference schemes, in W. Eckhaus & E. M. de Jager, eds, ‘Singular Perturbation Theory and Applications’, Springer Verlag.
- Lutz, M. (1996), *Programming Python*, O’Reilly and Associates.
- Nagumo, J., Arimoto, S. & Yoshizawa, S. (1962), ‘An active pulse transmission line simulating nerve axon’, *Proc. IRE* **50**, 2061–2070.
- Oldeman, B. E., Champneys, A. R. & B., K. (2003), ‘Homoclinic branch switching: A numerical implementation of Lin’s method’, *Int. J. of Bifurcation and Chaos* **10**, 2977–2999.
- Rodríguez-Luis, A. J. (1991), Bifurcaciones multiparamétricas en osciladores autónomos, PhD thesis, Department of Applied Mathematics, University of Seville, Spain.
- Rucklidge, A. & Mathews, P. (1995), ‘Analysis of the shearing instability in nonlinear convection and magnetoconvection’. Submitted to *Nonlinearity*.
- Russell, R. D. & Christiansen, J. (1978), ‘Adaptive mesh selection strategies for solving boundary value problems’, *SIAM J. Numer. Anal.* **15**, 59–80.
- Sandstede, B. (1993), Verzweigungstheorie homokliner Verdopplungen, PhD thesis, Universität Stuttgart.
- Sandstede, B. (1995a), Constructing dynamical systems possessing homoclinic bifurcation points of codimension two, In preparation.
- Sandstede, B. (1995b), Convergence estimates for the numerical approximation of homoclinic solutions, In preparation.
- Sandstede, B. (1995c), Numerical computation of homoclinic flip-bifurcations, In preparation.
- Scheffer, M. (1995), ‘Personal communication’.
- Smith, B., Boyle, J., Dongarra, J., Garbow, B., Ikebe, Y., Klema, X. & Moler, C. (1976), *Matrix Eigensystem Routines : EISPACK Guide*, Vol. 6, Springer Verlag.
- Uppal, A., Ray, W. H. & Poore, A. B. (1974), ‘On the dynamic behaviour of continuous stirred tank reactors’, *Chem. Eng. Sci.* **29**, 967–985.
- Wang, X. J. (1994), ‘Parallelization and graphical user interface of AUTO94’. M. Comp. Sci. Thesis, Concordia University, Montreal, Canada.

- Wang, X. J. & Doedel, E. J. (1995), AUTO94P : An experimental parallel version of AUTO, Technical report, Center for Research on Parallel Computing, California Institute of Technology, Pasadena CA 91125. CRPC-95-3.
- Yew, A. C. (2001), ‘Multipulses of nonlinearly-coupled Schrödinger equations’, *Journal of Differential Equations* **173**(1), 92–137.