AUTO 2000 : CONTINUATION AND BIFURCATION SOFTWARE FOR ORDINARY DIFFERENTIAL EQUATIONS (with HomCont)

Eusebius J. Doedel [1]
California Institute of Technology
Pasadena, California USA

Randy C. Paffenroth
California Institute of Technology
Pasadena, California USA

Alan R. Champneys
University of Bristol
United Kingdom

Thomas F. Fairgrieve
Ryerson Polytechnic University
Toronto, Canada

Yuri A. Kuznetsov
Universiteit Utrecht
The Netherlands

Bart E. Oldeman
University of Bristol
United Kingdom

Björn Sandstede
Ohio State University
Columbus, Ohio USA

Xianjun Wang
Concordia University
Montreal, Canada

July 30, 2002

[1] On leave from Concordia University, Montreal, Canada

Contents

1 Installing AUTO . 9

1.1 Typographical Conventions . . . 9
1.2 Installation. . . . 9
1.3 Restrictions on Problem Size. . . . 10
1.4 Compatibility with Older Versions. . . . 11
1.5 Parallel Version. . . . 11

2 Overview of Capabilities. 13

2.1 Summary. . . . 13
2.2 Algebraic Systems. . . . 13
2.3 Ordinary Differential Equations. . . . 14
2.4 Parabolic PDEs. . . . 15
2.5 Discretization. . . . 16

3 How to Run AUTO . 17

3.1 User-Supplied Files. . . . 17
3.1.1 The equations-file xxx.c . . . 17
3.1.2 The constants-file c.xxx . . . 17
3.2 User-Supplied Subroutines. . . . 18
3.3 Arguments of **stpnt**. . . . 18
3.4 User-Supplied Derivatives. . . . 19
3.5 Output Files. . . . 19

4 Command Line User Interface. 21

4.1 Typographical Conventions . . . 21
4.2 General Overview. . . . 21
4.3 First Example . . . 22
4.4 Scripting . . . 25
4.5 Second Example . . . 26
4.6 Extending the AUTO 2000 CLUI . . . 29
4.7 Bifurcation Diagram Files . . . 31
4.8 Solution Files . . . 31
4.9 The .autorc File . . . 36
4.10 Two Dimensional Plotting Tool . . . 36
4.11 Three Dimensional Plotting Tool . . . 38

4.12 Quick Reference 39
4.13 Reference 41
4.13.1 commandAppend 41
4.13.2 commandCat 41
4.13.3 commandCd 42
4.13.4 commandClean 42
4.13.5 commandCopyAndLoadDemo 43
4.13.6 commandCopyDataFiles 43
4.13.7 commandCopyDemo 44
4.13.8 commandCopyFortFiles 44
4.13.9 commandCreateGUI 45
4.13.10 commandDeleteDataFiles 45
4.13.11 commandDeleteFortFiles 46
4.13.12 commandDouble 46
4.13.13 commandInteractiveHelp 47
4.13.14 commandLs 47
4.13.15 commandMoveFiles 48
4.13.16 commandParseConstantsFile 48
4.13.17 commandParseDiagramAndSolutionFile 49
4.13.18 commandParseDiagramFile 49
4.13.19 commandParseSolutionFile 50
4.13.20 commandPlotter 50
4.13.21 commandPlotter3D 51
4.13.22 commandQueryBranchPoint 51
4.13.23 commandQueryEigenvalue 52
4.13.24 commandQueryFloquet 52
4.13.25 commandQueryHopf 53
4.13.26 commandQueryIterations 53
4.13.27 commandQueryLimitpoint 54
4.13.28 commandQueryNote 54
4.13.29 commandQuerySecondaryPeriod 55
4.13.30 commandQueryStepsize 55
4.13.31 commandRun 56
4.13.32 commandRunnerConfigFort2 56
4.13.33 commandRunnerLoadName 57
4.13.34 commandRunnerPrintFort2 58
4.13.35 commandShell 58
4.13.36 commandTriple 59
4.13.37 commandUserData 59
4.13.38 commandWait 60

5 Description of AUTO -Constants. 61
5.1 The AUTO -Constants File. 61
5.2 Problem Constants. 61
5.2.1 `NDIM` 61

5.2.2 NBC 61
5.2.3 NINT 62
5.2.4 JAC 62
5.3 Discretization Constants. 62
5.3.1 NTST 62
5.3.2 NCOL 62
5.3.3 IAD 62
5.4 Tolerances. 63
5.4.1 EPSL 63
5.4.2 EPSU 63
5.4.3 EPSS 63
5.4.4 ITMX 63
5.4.5 NWTN 63
5.4.6 ITNW 63
5.5 Continuation Step Size. 64
5.5.1 DS 64
5.5.2 DSMIN 64
5.5.3 DSMAX 64
5.5.4 IADS 64
5.5.5 NTHL 65
5.5.6 NTHU 65
5.6 Diagram Limits. 65
5.6.1 NMX 65
5.6.2 RL0 65
5.6.3 RL1 66
5.6.4 A0 66
5.6.5 A1 66
5.7 Free Parameters. 66
5.7.1 NICP, ICP 66
5.7.2 Fixed points. 66
5.7.3 Periodic solutions and rotations. 66
5.7.4 Folds and Hopf bifurcations. 67
5.7.5 Folds and period-doublings. 67
5.7.6 Boundary value problems. 67
5.7.7 Boundary value folds. 67
5.7.8 Optimization problems. 68
5.7.9 Internal free parameters. 68
5.7.10 Parameter overspecification. 68
5.8 Computation Constants. 69
5.8.1 ILP 69
5.8.2 ISP 69
5.8.3 ISW 69
5.8.4 MXBF 70
5.8.5 IRS 70
5.8.6 IPS 70

5.9 Output Control. 72
5.9.1 `NPR` 72
5.9.2 `IID` 72
5.9.3 `IPLT` 73
5.9.4 `NUZR` 74

6 Notes on Using AUTO . 75
6.1 Restrictions on the Use of `PAR`. 75
6.2 Efficiency. 75
6.3 Correctness of Results. 76
6.4 Bifurcation Points and Folds. 76
6.5 Floquet Multipliers. 76
6.6 Memory Requirements. 77

7 AUTO Demos : Tutorial. 78
7.1 Introduction. 79
7.2 ab : A Tutorial Demo. 79
7.3 Copying the Demo Files. 79
7.4 Executing all Runs Automatically. 80
7.5 Executing Selected Runs Automatically. 82
7.6 Using AUTO -Commands. 82
7.7 Plotting the Results with AUTO 83
7.8 Following Folds and Hopf Bifurcations. 84
7.9 Relabeling Solutions in the Data-Files. 86
7.10 Plotting the 2-Parameter Diagram. 86

8 AUTO Demos : Fixed points. 88
8.1 enz : Stationary Solutions of an Enzyme Model. 88
8.2 dd2 : Fixed Points of a Discrete Dynamical System. 89

9 AUTO Demos : Periodic solutions. 90
9.1 lrz : The Lorenz Equations. 91
9.2 abc : The A $\rightarrow$ B $\rightarrow$ C Reaction. 92
9.3 pp2 : A 2D Predator-Prey Model. 93
9.4 lor : Starting an Orbit from Numerical Data. 94
9.5 frc : A Periodically Forced System. 95
9.6 ppp : Continuation of Hopf Bifurcations. 96
9.7 plp : Fold Continuation for Periodic Solutions. 97
9.8 pp3 : Period-Doubling Continuation. 98
9.9 tor : Detection of Torus Bifurcations. 99
9.10 pen : Rotations of Coupled Pendula. 100
9.11 chu : A Non-Smooth System (Chua's Circuit). 102
9.12 phs : Effect of the Phase Condition. 103
9.13 ivp : Time Integration with Euler's Method. 104

10 AUTO Demos : BVP. 105
10.1 exp : Bratu's Equation. 105
10.2 int : Boundary and Integral Constraints. 106
10.3 bvp : A Nonlinear ODE Eigenvalue Problem. 107
10.4 lin : A Linear ODE Eigenvalue Problem. 108
10.5 non : A Non-Autonomous BVP. 109
10.6 kar : The Von Karman Swirling Flows. 110
10.7 spb : A Singularly-Perturbed BVP. 111
10.8 ezp : Complex Bifurcation in a BVP. 112

11 AUTO Demos : Parabolic PDEs. 113
11.1 pd1 : Stationary States (1D Problem). 114
11.2 pd2 : Stationary States (2D Problem). 115
11.3 wav : Periodic Waves. 116
11.4 brc : Chebyshev Collocation in Space. 117
11.5 brf : Finite Differences in Space. 118
11.6 bru : Euler Time Integration (the Brusselator). 119

12 AUTO Demos : Optimization. 120
12.1 opt : A Model Algebraic Optimization Problem. 121
12.2 ops : Optimization of Periodic Solutions. 122
12.3 obv : Optimization for a BVP. 126

13 AUTO Demos : Connecting orbits. 128
13.1 fsh : A Saddle-Node Connection. 129
13.2 nag : A Saddle-Saddle Connection. 130
13.3 stw : Continuation of Sharp Traveling Waves. 131

14 AUTO Demos : Miscellaneous. 133
14.1 pvl : Use of the Subroutine **pvls**. 134
14.2 ext : Spurious Solutions to BVB. 135
14.3 tim : A Test Problem for Timing AUTO 136

15 HomCont. 137
15.1 Introduction. 137
15.2 HomCont Files and Subroutines. 137
15.3 HomCont-Constants. 138
15.3.1 `NUNSTAB` 138
15.3.2 `NSTAB` 138
15.3.3 `IEQUIB` 138
15.3.4 `ITWIST` 139
15.3.5 `ISTART` 139
15.3.6 `NREV, IREV` 140
15.3.7 `NFIXED, IFIXED` 140
15.3.8 `NPSI, IPSI` 140

15.4 Restrictions on HomCont Constants. 140
15.5 Restrictions on the Use of `PAR`. 141
15.6 Test Functions. 141
15.7 Starting Strategies. 142
15.8 Notes on Running HomCont Demos. 144

16 HomCont Demo : san. 145
16.1 Sandstede's Model. 145
16.2 Inclination Flip. 145
16.3 Non-orientable Resonant Eigenvalues. 147
16.4 Orbit Flip. 147
16.5 Detailed AUTO -Commands. 149

17 HomCont Demo : mtn. 152
17.1 A Predator-Prey Model with Immigration. 152
17.2 Continuation of Central Saddle-Node Homoclinics. 152
17.3 Switching between Saddle-Node and Saddle Homoclinic Orbits. 154
17.4 Three-Parameter Continuation. 155
17.5 Detailed AUTO -Commands. 156

18 HomCont Demo : kpr. 159
18.1 Koper's Extended Van der Pol Model. 159
18.2 The Primary Branch of Homoclinics. 159
18.3 More Accuracy and Saddle-Node Homoclinic Orbits. 163
18.4 Three-Parameter Continuation. 166
18.5 Detailed AUTO -Commands. 167

19 HomCont Demo : cir. 169
19.1 Electronic Circuit of Freire *et al.* 169
19.2 Detailed AUTO -Commands. 172

20 HomCont Demo : she. 173
20.1 A Heteroclinic Example. 173
20.2 Detailed AUTO -Commands. 175

21 HomCont Demo : rev. 177
21.1 A Reversible System. 177
21.2 An R_1-Reversible Homoclinic Solution. 177
21.3 An R_2-Reversible Homoclinic Solution. 178
21.4 Detailed AUTO -Commands. 182

22 HomCont Demo : Homoclinic branch switching. 183
22.1 Branch switching at an inclination flip in Sandstede's model. 183
22.2 Branch switching for a Shil'nikov type homoclinic orbit in the FitzHugh-Nagumo equations. 190

22.3 Branch switching to a 3-homoclinic orbit in a 5th-order Korteweg-De Vries model 193

A Running AUTO using Command Mode. 196

A.0.1 Basic commands. 196
A.0.2 Plotting commands. 197
A.0.3 File-manipulation. 197
A.0.4 Diagnostics. 197
A.0.5 File-editing. 198
A.0.6 File-maintenance. 198
A.0.7 HomCont commands. 199
A.0.8 Copying a demo. 199
A.0.9 Pendula animation. 199
A.0.10 Viewing the manual. 199

B The Graphics Program PLAUT. 200

B.1 Basic PLAUT-Commands. 200
B.2 Default Options. 201
B.3 Other PLAUT-Commands. 202
B.4 Printing PLAUT Files. 202

C Graphical User Interface. 203

C.1 General Overview. 203
C.1.1 The Menu bar. 203
C.1.2 The Define-Constants-buttons. 204
C.1.3 The Load-Constants-buttons. 204
C.1.4 The Stop- and Exit-buttons. 204
C.2 The Menu Bar. 204
C.2.1 Equations-button. 204
C.2.2 Edit-button. 204
C.2.3 Write-button. 204
C.2.4 Define-button. 205
C.2.5 Run-button. 205
C.2.6 Save-button. 205
C.2.7 Append-button. 205
C.2.8 Plot-button. 205
C.2.9 Files-button. 205
C.2.10 Demos-button. 206
C.2.11 Misc.-button. 206
C.2.12 Help-button. 206
C.3 Using the GUI. 206
C.4 Customizing the GUI. 206
C.4.1 Print-button. 206
C.4.2 GUI colors. 207
C.4.3 On-line help. 207

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). For a description of the basic algorithms see Doedel, Keller & Kernévez (1991*a*), Doedel, Keller & Kernévez (1991*b*). This version of AUTO incorporates the HomCont algorithms of Champneys & Kuznetsov (1994), Champneys, Kuznetsov & Sandstede (1996) for the bifurcation analysis of homoclinic orbits. The graphical user interface was written by Wang (1994). The Floquet multiplier algorithms were written by Fairgrieve (1994), Fairgrieve & Jepson (1991).

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. 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 and the pendula animation program. An earlier graphical user interface for AUTO on SGI machines was written by Taylor & Kevrekidis (1989). 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. Work on this updated version was supported by a general research grant from NSERC (Canada).

The development of HomCont has much benefitted from various pieces of 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 and C. J. Budd, University of Bristol. Financial support for collaboration was received from the U.K. Engineering and Physical Science Research Council and the Nuffield Foundation.

Chapter 1

Installing AUTO .

1.1 Typographical Conventions

This manual uses the following conventions.

`command`	This font is used for commands which you can type in.
`PAR`	This font is used for AUTO parameters.
filename	This font is used for file and directory names.
variable	This font is used for environment variable.
site	This font is used for world wide web and ftp sites.
function	This font is used for function names.

1.2 Installation.

The AUTO files are available via HTTP from
http://www.ama.caltech.edu/~redrod/auto2000/distribution/.

bzipped Postscript manual	*auto2000-0.9.6.ps.bz2*
gzipped Postscript manual	*auto2000-0.9.6.ps.gz*
compressed Postscript manual	*auto2000-0.9.6.ps.Z*
tarred and gzipped source code	*auto2000-0.9.6.tgz*
tarred and bzipped source code	*auto2000-0.9.6.tbz2*
tarred and compressed source code	*auto2000-0.9.6.tar.Z*
zipped source code	*auto2000-0.9.6.zip*

Below it is assumed that you are using the Unix shell csh and that the file auto2000-0.9.6.tar.Z is in your main directory.

While in your main directory, enter the commands `uncompress auto2000-0.9.6.tar.Z`, followed by `tar xvfo auto2000-0.9.6.tar`. This will result in a directory auto, with one subdirectory, auto/2000. Type `cd auto/2000` to change directory to auto/2000. 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.

There is a new CLUI under development which includes some of the capabilities of the old GUI and will eventually be the recommend way to run AUTO. More information on the CLUI may be found in Chapter 4. The new CLUI does not require any additional options to be passed to the `configure` script.

To run the new Command Line User Interface (CLUI) and the old command language 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/2000/cmds/auto.env.sh`. On the other hand, if you are using a `csh` compatible shell, such as `csh` or `tcsh`, enter the command `source $HOME/auto/2000/cmds/auto.env.csh`.

There is an old and unsupported Graphical User Interface (GUI) which requires the X-Window system and Motif, and it is not compiled by default. Note that AUTO can be very effectively run in Command Mode, i.e., the GUI is not strictly necessary. To compile AUTO with the old GUI, type `configure --enable-gui` and then make in directory auto/2000.

The PostScript conversion command @ps will be enabled if the configure script detects the appropriate software, but you may have to enter the correct printer name in auto/2000/cmds/@pr.

To generate the on-line manual, type `make` in auto/2000/doc.

To prepare AUTO for transfer to another machine, type `make superclean` in directory auto/2000 before creating the tar-file. This will remove all executable, object, and other non-essential files, and thereby reduce the size of the package.

AUTO can be tested by typing `make > TEST &` in directory auto/2000/test. This will execute a selection of demos from auto/2000/demos and write a summary of the computations in the file TEST. The contents of TEST can then be compared to other test result files in directory auto/2000/test. Note that minor differences are to be expected due to architecture and compiler differences.

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)).

1.3 Restrictions on Problem Size.

There are size restrictions in the file auto/2000/src/auto_c.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 5 for the significance these constants. Their maxima are denoted by the corresponding constant followed by an X. For example, `NPARX` in auto_c.h denotes the maximum value of `NPAR`. If the maxima of `NBIF` is exceeded in an AUTO -run then a message will be printed. On the other hand, the maximum value of `NPAR`, if exceeded, may lead to unreported errors. Upon installation `NPARX`=36; it should never be decreased below that value; see also Section 6.1. Size restrictions can be changed by editing auto_c.h. This must be followed by recompilation by typing `make` in directory auto/2000/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 2 `NDIM`+1 for algebraic equations, and 2 `NDIM` for ordinary differential equations, respectively. Similarly, for the continuation of Hopf bifurcations, the effective dimension is 3 `NDIM`+2.

1.4 Compatibility with Older Versions.

There are two changes compared to early versions of AUTO 94 : The user-supplied equations-files must contain the subroutine **pvls**. For an example of use of **pvls** see the demo `pvl` in Section 14.1. There is also a small change in the `q.xxx` data-file. If necessary, older AUTO 94 files can be converted using the `@94to97` command; see Section A. Data files from AUTO 97 are fully compatible with AUTO 2000 , but as AUTO 2000 is written in C user defined function files from AUTO 97 , which are generally in Fortran, must be rewritten.

1.5 Parallel Version.

AUTO 2000 contains code which allows it to run in on various types of parallel computers. Namely, it can use either the Pthreads library for running on shared-memory multi-processors, or the MPI message passing library. When the `configure` script is run it will try to find the above two libraries, and if it is successful it will include their functionality into AUTO 2000 . To force the `configure` script not to use either of the above libraries, one may type `configure --without-mpi` or `configure --without-pthreads`, and then type `make`. One may even preclude both by typing `configure --without-mpi --without-pthreads` and then typing `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 2000 may easily be controlled, and even run in a serial mode, through the use of command line options at run time. The command line options are listed in Table 1.1.

-v	Give verbose output.
-m	Use the Message Passing Interface library for parallelization.
-t	Use the Pthreads library for parallelization. This option takes one of three arguments. **conpar** parallelizes the condensation of parameters routine. **setubv** parallelizes the Jacobian setup routine. **both** parallelizes both routines. In general the recommended option is 'both'.
-#	The number of processing units to use (currently only used with the -t option).

Table 1.1: Command line options.

For example, to run the AUTO 2000 executable `auto.exe` in serial mode you just type `auto.exe`.

To run the same command in parallel using the Pthreads library on 4 processors you type `auto.exe -t both -# 4`. If you were to try and run the above command on a machine which did not have the Pthreads library, the command would exit with an error and inform you that the Pthreads library is not available.

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 2000 on four processors, with the above external program, you would type `mpirun -np 4 auto.exe -m`. Please see your local MPI documentation for more detail. As with the Pthreads library, if you were to try and run the above command on a machine which did not have MPI, the command would exit with an error and inform you that MPI is not available.

The commands in the `auto/2000/cmds` directory and described in Chapter 3 may be used with the parallel version as well, by setting the *AUTO_COMMAND_PREFIX* and *AUTO_COMMAND_ARGS* environment variables. For example, to the run AUTO 2000 in parallel using the Pthreads library on 4 processors just type `setenv AUTO_COMMAND_ARGS ‘‘-t both -# 4’’` and then use the commands in `auto/2000/cmds` normally. To run AUTO 97 in parallel using the MPI library on 4 processors just type `setenv AUTO_COMMAND_ARGS ‘‘-m’’` and `setenv AUTO_COMMAND_PREFIX ‘‘mpirun -np 4’’`, and then use the commands in `auto/2000/cmds` normally. The previous examples assumed you are using the `csh` shell or the `tcsh` shell, for other shells you should modify the commands appropriately.

Chapter 2

Overview of Capabilities.

2.1 Summary.

AUTO can do a limited bifurcation analysis of algebraic systems

$$f(u,p)=0, \qquad f(\cdot,\cdot),u \in \mathrm{R}^n, \tag{2.1}$$

and of systems of ordinary differential equation (ODEs) of the form

$$u'(t)=f\big(u(t),p\big), \qquad f(\cdot,\cdot),u(\cdot) \in \mathrm{R}^n, \tag{2.2}$$

Here p denotes one or more free parameters.

It can also do certain stationary solution and wave calculations for the partial differential equation (PDE)

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

where D denotes a diagonal matrix of diffusion constants. The basic algorithms used in the package, as well as related algorithms, can be found in Keller (1977), Keller (1986), Doedel, Keller & Kernévez (1991*a*), Doedel, Keller & Kernévez (1991*b*).

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) the program can :

- Compute solution branches.
  (Demo `ab`; Run 1.)

- Locate branch points and automatically compute bifurcating branches.
  (Demo `pp2`; Run 1.)

- 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 branches and successively continue such extrema in more parameters.
  (Demo opt.)

2.3 Ordinary Differential Equations.

For the ODE (2.2) the program can :

- Compute branches of stable and unstable periodic solutions and compute the Floquet multipliers, that determine stability, along these branches. 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 branches 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 branch 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 branches 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 branches 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 branches of the related ODE

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

where $z = x - ct$, with the wave speed c specified by the user.
(Demo `wav`; Run 2.)

- Trace out branches 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 branch, 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 branches 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 branch 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

How to Run AUTO .

3.1 User-Supplied Files.

The user must prepare the two files described below. This can be done with the GUI described in Chapter 4, or independently.

3.1.1 The equations-file `xxx.c`

A source file `xxx.c` containing the C subroutines **func**, **stpnt**, **bcnd**, **icnd**, **fopt**, and **pvls**. Here `xxx` stands for a user-selected name. If any of these subroutines is irrelevant to the problem then its body need not be completed. Examples are in `auto/2000/demos`, where, e.g., the file `ab/ab.c` defines a two-dimensional dynamical system, and the file `exp/exp.c` defines a boundary value problem. The simplest way to create a new equations-file is to copy an appropriate demo file. In GUI mode, this file can be directly loaded with the GUI-button `Equations/New`; see Section C.2.

3.1.2 The constants-file `c.xxx`

AUTO -constants for `xxx.c` are normally expected in a corresponding file `c.xxx`. Specific examples include `ab/c.ab` and `exp/c.exp` in `auto/2000/demos`. See Chapter 5 for the significance of each constant.

3.2 User-Supplied Subroutines.

The purpose of each of the user-supplied subroutines in the file xxx.c is described below.

- **func** : defines the function $f(u,p)$ in (2.1), (2.2), or (2.3).
- **stpnt** : This subroutine is called only if `IRS`=0 (see Section 5.8.5 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 subroutine **bcnd** that defines the boundary conditions.
  (Demo exp, kar.)
- **icnd** : A subroutine **icnd** that defines the integral conditions.
  (Demos int, lin.)
- **fopt** : A subroutine **fopt** that defines the objective functional.
  (Demos opt, ops.)
- **pvls** : A subroutine **pvls** for defining "solution measures".
  (Demo pvl.)

3.3 Arguments of stpnt.

Note that the arguments of **stpnt** depend on the solution type :

- When starting from a fixed point or an analytically or numerically known space-dependent solution, **stpnt** must have four arguments, namely, (`NDIM,U,PAR,T`). Here T is the independent space variable which takes values in the interval $[0,1]$. T is ignored in the case of fixed points.
  (Demos exp and ab.)
- Similarly, when starting from an analytically known time-periodic solution or rotation, the arguments of **stpnt** are (`NDIM,U,PAR,T`), where T denotes the independent time variable which takes values in the interval $[0,1]$. In this case one must also specify the period in `PAR(11)`.
  (Demos frc, lor, pen.)
- When using the `@fc` command (Section A) for conversion of numerical data, **stpnt** must have four arguments, namely, (`NDIM,U,PAR,T`). In this case only the parameter values need to be defined in **stpnt**. (Demos lor and pen.)

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 5.2.4. 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.

3.5 Output Files.

AUTO writes four output-files :

- fort.6 : A summary of the computation is written in fort.6, which usually corresponds to the window in which AUTO is run. Only special, labeled solution points are noted, namely those listed in Table 3.1. 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 branch; normal termination
MX	(-9)	Abnormal termination; no convergence

Table 3.1: 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 inspected. The amount of diagnostic data can be controlled via the AUTO -constant `IID`; see Section 5.9.2.

The user has some control over the fort.6 (screen) and fort.7 output via the AUTO -constant `IPLT` (Section 5.9.3). Furthermore, the subroutine **pvls** can be used to define "solution measures" which can then be printed by “parameter overspecification”; see Section 5.7.10. For an example see demo pvl.

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

The graphics program PLAUT can be used to graphically inspect the data in fort.7 and fort.8; see Chapter B.

Chapter 4

Command Line User Interface.

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 2000 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 2000 CLUI.

4.2 General Overview.

The AUTO 2000 command line user interface (CLUI) is similar to the command language described in Section A 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 2000 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/2000 *or in any of its subdirectories.*) Then type

`auto`.

If your command search path has been correctly set (see Section 1.2), this command will start the AUTO 2000 CLUI interactive interpretor and provide you with the AUTO 2000 CLUI prompt.

```
> 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 2000 CLUI.

In addition to the examples in the following sections there are several example scripts which can be found in auto/2000/demos/python and are listed in Table 4.1. These scripts are fully annotated and provide good examples of how AUTO 2000 CLUI scripts are written. The scripts in auto/2000/demos/python/n-body are espcially lucid examples and preform various related parts of a calculation involving the gravitional 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 show 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 2000 CLUI. In this example we copy the ab demo from the AUTO 2000 installation directory and run it. For more information on the ab demo see Section 7.2. 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 4.13.7 in the reference) copies the files in $AUTO_DIR/demo/ab into the work directory.

Next, `load(equation='ab')` (Section 4.13.33 in the reference) informs the AUTO 2000 CLUI that the name of the user defined function file is ab.c. The command `load` is one of the most commonly used commands in the AUTO 2000 CLUI, since it reads and parses the user files which are manipulated by other commands. The AUTO 2000 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 A.

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 2000 unless it is loaded in again after the changes are made.*

Finally, `run()` (Section 4.13.31 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 2000 .

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 3.1). There are two other files types that can be read in using the `load` command, and they are the restart solution file (Section 3.5) and the HomCont parameter file (Section 15.2).

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 4.11.
userScript.py	The loadable expert demo script from Figure 4.12.
fullTest.auto	A script which uses the entire AUTO 2000 command set, except for the plotting commands.
plotter.auto	A demonstration of some of the plotting capabilities of AUTO 2000 .
fullTest.auto	A script which implements the tutorial from Section 7.2.
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 2000 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 2000 CLUI.

Unix-COMMAND	ACTION
`auto`	start the AUTO 2000 CLUI
AUTO 2000 CLUI COMMAND	**ACTION**
`copydemo('ab')`	copy the demo files to the work directory
`load(equation='ab')`	load the filename `ab.c` into memory
`load(constants='ab.1')`	load the contents of the file `r.ab.1` into memory
`run()`	run AUTO 2000 with the current set of files

Table 4.2: Running the demo `ab` files.

```
> 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> copydemo('ab')
Copying demo ab ... done
AUTO> load(equation='ab')
Runner configured
AUTO> load(constants='ab.1')
Runner configured
AUTO> run()
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    9.502E-02
ab ... done
AUTO>
```

Figure 4.2: Typing `auto` at the Unix shell prompt starts the AUTO 2000 CLUI. The rest of the commands are interpreted by the AUTO 2000 CLUI.

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 2000 CLUI into the standard names used by AUTO 2000 . 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 2000 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.c` 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 2000 CLUI prompt, but since the AUTO 2000 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 where 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 2000 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 2000 CLUI script, in this Section we will describe a more complex example, which introduces several new AUTO 2000 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 2000 CLUI commands. First, `sv('bvp')` (Section 4.13.6 in the reference) saves the results of the AUTO 2000 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 2000 for further calculations.

Up to this point all of the commands presented have had analogs in the command language discussed in Section A, and the AUTO 2000 CLUI has been designed in this way to make it easy for users to migrate from the old command language to the AUTO 2000 CLUI. The next command, namely `data = sl('bvp')` (Section 4.13.19 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 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 4.13.32 in the reference) changes the `NTST` value to `50` (see Section 5.2.1). To be precise, the command `ch("NTST",50)`

```
> 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 2000 CLUI by typing auto demo1.auto and the output is shown.

only modifies the "in memory" version of the AUTO 2000 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 2000 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 2000 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 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 condi-

```
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 2000 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 2000 CLUI script.

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

Figure 4.9: The second part of the complex AUTO 2000 CLUI script.

tional. 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 2000 constants file to set `IRS` (see Section 5.8.5) equal to the label of the bifurcation point. We then use `ch("ISW", -1)` to change the AUTO 2000 constant `ISW` to `-1`, which indicates a branch switch (see Section 5.8.3).

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 4.13.1 in the reference). In addition, as can be seen in Figure 4.10, the `#` character is the Python comment character. When the Python interpretor encounters a `#` character it ignores everything from that character to the end of the line.

Finally, we us `ch("DS",-pr("DS"))` to change the AUTO 2000 initial step size from positive to negative, which allows us to compute the bifurcating branch in the other direction (see Section 5.5.1). Running the AUTO 2000 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 2000 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 4.13.20 in the reference), and the command `wait()` causes the AUTO 2000 CLUI to wait for input. You may now exit the AUTO 2000 CLUI by pressing any key in the window in which you started the AUTO 2000 CLUI.

4.6 Extending the AUTO 2000 CLUI

The code in Figure 4.7 performed a very useful and common procedure, it started an AUTO 2000 calculation and performed additional continuations at every point which AUTO 2000 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 2000 CLUI. Several examples of such “expert” scripts can be found in auto/2000/demos/python/n-body.

Just as loops and conditionals can be used in Python , one can also define functions. For example, Figure 4.11 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 2000 CLUI commands, and must be included in all AUTO 2000 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.

```
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 4.11: This Figure shows a complex AUTO 2000 CLUI script written as a function. The source for this script can be found in $AUTO_DIR/demos/python/userScript.xauto.

While the script in Figure 4.11 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 2000 CLUI and used to create a new command, as in Figure 4.12. First, we create a file called userScript.py which contains the script from Figure 4.11, 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 2000 CLUI with the Unix command `auto`, and once the AUTO 2000 CLUI is running we use the command `from userScript import *`, to import the file userScript.py into the AUTO 2000 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 2000 CLUI, for example by typing `myRun('bvp')`.

4.7 Bifurcation Diagram Files

Using the `commandParseDiagramFile` command (Section 4.13.18 in the reference) the user can parse and read into memory an AUTO 2000 bifurcation diagram file. For example, the command `commandParseDiagramFile('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 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 4.13, 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)`.

This class has two methods that are particularily useful for creating data which can be used in other programs. First, there is a method called `toArray` which takes a bifurcation diagram 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. For example, we again assume that the parsed object is contained in a variable `data`. If one wanted to have the bifurcation diagram returned as a Python array one would type `data.toArray()`. Similarily, if one wanted to write out the bifurcation diagram to the file `outputfile` one would type `data.writeRawFilename('outputfile')`.

4.8 Solution Files

Using the `commandParseSolutionFile` command (Section 4.13.19 in the reference) the user can parse and read into memory an AUTO 2000 bifurcation solution file. For example, the command `commandParseSolutionFile('ab')` would parse the file b.ab (if you are using the standard file-

```
> cp \$AUTO\_DIR/python/demo/userScript.py .
> ls
userScript.py
> cat userScript.py
# This is an example script for the AUTO2000 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 4.12: This Figure shows the functional version of the AUTO 2000 CLUI from Figure 4.11 being used as an extension to the AUTO 2000 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 4.13: 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.

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 2000 output.

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

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 invarient 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.

name 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 4.14, 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 4.14: 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, sssume 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)`.

This class has two methods that are particularily useful for creating data which can be used in other programs. First, there is a method called `toArray` which takes a solution and returns a standard Python array. Second, there is a method called `writeRawFilename` which will create a standard ASCII file which contains the solution. 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. For example, we again assume that the parsed object is contained in a variable `data`. If one wanted to have the solution with label `57` returned as a Python array one would type

Query string	Meaning
data	An array which contains the AUTO 2000 output.
Branch number	The number of the branch to which the solution belongs.
ISW	The ISW value used to start the calcluation. See Section 5.8.3.
Label	The label of the solution.
NCOL	The number of collocation points used to compute the solution. See Section 5.3.2.
NTST	The number of mesh intervals used to compute the solution. See Section 5.3.1.
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.

`data(57).toArray()`. Similarily, if one wanted to write out the solution to the file `outputfile` one would type `data(57).writeRawFilename('outputfile')`.

4.9 The .autorc File

Much of the default behavior of the AUTO 2000 CLUI can be controlled by the .autorc file. The .autorc file can exist in either the main AUTO 2000 directory, the users home directory, or the current directory. For any option which is defined in more then 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 2000 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 2000 CLUI commands. Each line thereafter is a definition of a command, similiar to `branchPoint =commandQueryBranchPoint`. The right hand side of the assignment is the internal AUTO 2000 CLUI name for the command, while the left hand side is the desired alias. Aliases and internal names may be used interchangably, 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 4.13.

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

4.10 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 plote 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 decribed in Table 4.7. In addition, the options can be set from within the CLUI. For example, the set of commands in Figure 4.15 shows how to create a plotter and change its background color to black. The demo script auto/2000/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 4.15: This example shows how a plotter is created, and how the background color may be changed to black. All other configuration options are set similarily. Note, the above commands assume that the files fort.7 and fort.8 exist in the current directory.

Query string	Meaning
background	The background color of the plot.
bifurcation_column_defaults	A set of bifurcation columns the user is likely to use.
bifurcation_diagram	A parsed bifurcation diagram file to plot.
bifurcation_diagram_filename	The filename of the bifurcation diagram to plot.
bifurcation_symbol	The symbol to use for bifurcation points.
bifurcation_x	The column to plot along the X-axis for bifurcation diagrams.
bifurcation_y	The column to plot along the Y-axis for bifurcation diagrams.
color_list	A list of colors to use for multiple plots.
decorations	Turn on or off the axis, tick marks, etc.
error_symbol	The symbol to use for error points.
foreground	The background color of the plot.
grid	Turn on or off the grid.
hopf_symbol	The symbol to use for Hopf bifurcation points.
index	An array of indicies 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.
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”.
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 2000 CLUI two dimensional plotting window and their meanings.

4.11 Three Dimensional Plotting Tool

NOTE: the documentation in this section is under developement.

The AUTO 2000 three dimensional plotting tool can use DataViewer or OpenInventor for rendering three dimensional representations of bifurcation diagrams and solutions and is under active development. Neither DataViewer nor OpenInventor are provided with AUTO 2000 and must be downloaded seperately. If you are interested in the three dimensional plotting tool please contact *redrod@acm.org*.

4.12 Quick Reference

In this section we have created a table of all of the AUTO 2000 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
commandAppend	ap append	Append data files.
commandCat	cat	Print the contents of a file
commandCd	cd	Change directories.
commandClean	clean cl	Clean the current directory.
commandCopyAndLoadDemo	dm demo	Copy a demo into the current directory and load it.
commandCopyDataFiles	copy cp	Copy data files.
commandCopyDemo	copydemo	Copy a demo into the current directory.
commandCopyFortFiles	sv save	Save data files.
commandCreateGUI	gui	Show AUTOs graphical user interface.
commandDeleteDataFiles	delete dl	Delete data files.
commandDeleteFortFiles	df deletefort	Clear the current directory of fort files.
commandDouble	double db	Double a solution.
commandInteractiveHelp	man help	Get help on the AUTO commands.
commandLs	ls	List the current directory.
commandMoveFiles	move mv	Move data-files to a new name.
commandParseConstantsFile	cn constantsget	Get the current continuation constants.
commandParseDiagramAndSolutionFile	bt diagramandsolutionget	Parse both bifurcation diagram and solution.
commandParseDiagramFile	dg diagramget	Parse a bifurcation diagram.
commandParseSolutionFile	sl solutionget	Parse solution file:
commandPlotter	p2 pl plot	2D plotting of data.
commandPlotter3D	plot3 p3	3D plotting of data.
commandQueryBranchPoint	br bp branchpoint	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 interations.
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.
commandRunnerLoadName	ld load	Load files into the AUTO runner.
commandRunnerPrintFort2	pc pr printconstant	Print continuation parameters.
commandShell	shell	Run a shell command.
commandTriple	tr triple	Triple a solution.
commandUserData	us userdata	Covert user-supplied data files.
commandWait	wait	Wait for the user to enter a key.

4.13 Reference

4.13.1 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

4.13.2 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

4.13.3 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

4.13.4 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

4.13.5 commandCopyAndLoadDemo

Purpose

Copy a demo into the current directory and load it.

Description

Type commandCopyAndLoadDemo('xxx') to copy all files from auto/2000/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

4.13.6 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

4.13.7 commandCopyDemo

Purpose

Copy a demo into the current directory.

Description

Type commandCopyDemo('xxx') to copy all files from auto/2000/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

4.13.8 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

4.13.9 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

4.13.10 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

4.13.11 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 deletefort

4.13.12 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

4.13.13 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

4.13.14 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

4.13.15 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

4.13.16 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

4.13.17 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

4.13.18 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

4.13.19 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

4.13.20 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

4.13.21 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

4.13.22 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

4.13.23 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

4.13.24 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

4.13.25 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

4.13.26 commandQueryIterations

Purpose

Print the number of Newton interations.

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

4.13.27 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

4.13.28 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

4.13.29 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

4.13.30 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

4.13.31 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).
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').

Aliases

r run rn

4.13.32 commandRunnerConfigFort2

Purpose

Modify continuation constants.

Description

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

Aliases

changeconstant cc ch

4.13.33 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
-------------------------------------------
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: 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').

Aliases

ld load

4.13.34 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

4.13.35 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

4.13.36 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

4.13.37 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(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' 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

4.13.38 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

Description of AUTO -Constants.

5.1 The AUTO -Constants File.

As described in Section 3.1, if the equations-file is xxx.c 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.ab in directory auto/2000/demos/ab is listed below. (The tutorial demo ab is described in detail in Chapter 7.)

```
2 1 0 1               NDIM,IPS,IRS,ILP
1   1                 NICP,(ICP(I),I=1,NICP)
50 4 3 1 1 0 0 0      NTST,NCOL,IAD,ISP,ISW,IPLT,NBC,NINT
100 0. 0.15 0. 100.   NMX,RL0,RL1,A0,A1
100 10 2 8 5 3 0      NPR,MXBF,IID,ITMX,ITNW,NWTN,JAC
1.e-6 1.e-6 0.0001    EPSL,EPSU,EPSS
0.01 0.005 0.05 1     DS,DSMIN,DSMAX,IADS
1                     NTHL,((I,THL(I)),I=1,NTHL)
11 0.
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.

5.2 Problem Constants.

5.2.1 NDIM

Dimension of the system of equations as specified in the user-supplied subroutine **func**.

5.2.2 NBC

The number of boundary conditions as specified in the user-supplied subroutine **bcnd**.
(Demos exp, kar.)

5.2.3 NINT

The number of integral conditions as specified in the user-supplied subroutine **icnd**.
(Demos int, lin, obv.)

5.2.4 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 subroutines **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.

  (Demos int, dd2, obt, plp, ops.)

  (For ISW see Section 5.8.3.)

5.3 Discretization Constants.

5.3.1 NTST

The number of mesh intervals used for discretization. NTST remains fixed during any particular run, but can be changed when restarting. Recommended value of NTST : As small as possible to maintain convergence.
(Demos exp, ab, spb.)
(For mesh adaption see IAD in Section 5.3.3.)

5.3.2 NCOL

The number of Gauss collocation points per mesh interval, ($2 \leq$ NCOL ≤ 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.

5.3.3 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 branch. 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 branch with a fixed mesh (`IAD=0`). Be sure to set `IAD` back to `IAD=3` for computing eventual non-trivial bifurcating solution branches.

5.4 Tolerances.

5.4.1 EPSL

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

5.4.2 EPSU

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

5.4.3 EPSS

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

5.4.4 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 ITMX=8, used in most demos.

5.4.5 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 NWTN+1 to ITNW. The choice 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.

5.4.6 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 branch is discontinued and a message printed in fort.9. The recommended value is ITNW=5, but ITNW=7 may be used for "difficult" problems, for example, demos spb, chu, plp, etc.

5.5 Continuation Step Size.

5.5.1 `DS`

AUTO uses pseudo-arclength continuation for following solution branches. The pseudo-arclength stepsize is the distance between the current solution and the next solution on a branch. 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 branch. (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 `DSMIN` $\leq |$ $|DS \leq$ `DSMAX` must be satisfied. The precise choice of `DS` is problem-dependent; the demos use a value that was found appropriate after some experimentation.

5.5.2 `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 6.2.

5.5.3 `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 6.2.

5.5.4 `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 branch 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 $|$ $|DS$ 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 $|$ $|DS$ reaches `DSMIN`. In the latter case non-convergence will be signalled. The strongly recommended value is `IADS`=1, which is used in almost all demos.

5.5.5 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.

5.5.6 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.

5.6 Diagram Limits.

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

- 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 5.9.4.

5.6.1 NMX

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

5.6.2 RL0

The lower bound on the principal continuation parameter. (This is the parameter which appears first in the ICP list; see Section 5.7.1.).

5.6.3 `RL1`

The upper bound on the principal continuation parameter.

5.6.4 `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 5.9.3 for choosing another principal solution measure.)

5.6.5 `A1`

The upper bound on the principal solution measure.

5.7 Free Parameters.

5.7.1 `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.

5.7.2 Fixed points.

The simplest case is the continuation of a solution branch to the system $f(u,p) = 0$, where $f(\cdot,\cdot), u \in \mathrm{R}^n$, 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.

5.7.3 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 addd `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.

5.7.4 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.

5.7.5 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)`.

5.7.6 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 branch. 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)`.

5.7.7 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.

5.7.8 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.

5.7.9 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.

5.7.10 Parameter overspecification.

The number of specified parameter indices is allowed to be 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 subroutine 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 pvl. 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 subroutine 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.pvl 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.

5.8 Computation Constants.

5.8.1 `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.

2

5.8.2 `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 6.4.

- `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. The Floquet multipliers will be output to the diagnostic file.

5.8.3 `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 5.7.

5.8.4 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 branches will not be computed. If MXBF is positive then the bifurcating branches of the first MXBF branch points will be traced out in both directions. If MXBF is negative then the bifurcating branches of the first | MXBF | branch points will be traced out in only one direction.

5.8.5 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 subroutine stpnt; see also Section 3.3. For representative examples of analytical starting solutions see demos ab and frc. For starting from unlabeled numerical data see the *@fc* command (Section A) 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 q.xxx; see also the *@r* and *@R* commands in Section A. Various AUTO -constants can be modified when restarting.

5.8.6 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 subroutine **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 subroutine **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 subroutine **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 subroutine **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 15-21 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 subroutine **func**), initialize the wave speed in PAR(10), initialize the diffusion constants in PAR(15,16,$\cdots$), 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,$\cdots$). 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 subroutine **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 subroutine **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.

5.9 Output Control.

5.9.1 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$\geq$ 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 5.9.4. For these reasons, and to limit the output volume, it is recommended that NPR output be kept to a minimum.

5.9.2 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 (1991*b*).

- 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.

5.9.3 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 5.5.5 and Section 5.5.6) affect the definition of the L_2-norm.

- If $0 <$ IPLT $\leq$ NDIM then the maximum of the IPLT'th solution component is printed.

- If $-$ NDIM $\leq$ IPLT <0 then the minimum of the IPLT'th solution component is printed. (Demo fsh.)

- If NDIM $<$ IPLT $\leq 2^*$ NDIM then the integral of the (IPLT$-$ NDIM)'th solution component is printed. (Demos exp, lor.)

- If 2^* NDIM $<$ IPLT $\leq 3^*$ NDIM then the L_2-norm of the (IPLT$-$ 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 subroutine **pvls** provides a second, more general way of defining solution measures; see Section 5.7.10.

5.9.4 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 pvl. For details on overspecified parameters see Section 5.7.10.

Chapter 6

Notes on Using AUTO .

6.1 Restrictions on the Use of PAR.

The array `PAR` in the user-supplied subroutines 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 5.7. The following restrictions apply :

- The maximum number of parameters, `NPARX` in auto/2000/src/auto_c.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.

6.2 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 5.4 and Section 5.5. Finally, it is recommended that initial computations be done with ILP=0; no fold detection; and ISP=1; no bifurcation detection for ODEs.

6.3 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 branch typically serves as a warning. Repeating the computation with increased NTST is then recommended.

6.4 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 branch 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.

6.5 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 3.1) have indeed been correctly classified.

6.6 Memory Requirements.

Pre-defined maximum values of certain AUTO -constants are in `auto/2000/src/auto_c.h`; see also Section 1.3. These maxima affect the run-time memory requirements and should not be set to unnecessarily large values. If an application only solves algebraic systems and if `NDIM` is “large” then memory requirements can be much reduced by setting each of `NTSTX`, `NCOLX`, `NBCX`, `NINTX`, equal to 1 in `auto/2000/src/auto_c.h`, followed by recompilation of the AUTO libraries.

Chapter 7

AUTO Demos : Tutorial.

7.1 Introduction.

The directory auto/2000/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.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 ab in detail. A brief description of other demos is given in later chapters.

7.2 ab : A Tutorial Demo.

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

$$\begin{aligned} u_1' &= -u_1 + p_1(1-u_1)e^{u_2}, \\ u_2' &= -u_2 + p_1 p_2 (1-u_1)e^{u_2} - p_3 u_2. \end{aligned} \tag{7.1}$$

7.3 Copying the Demo Files.

The commands listed in Table 7.1 will copy the demo files to your work directory.

Unix-COMMAND	ACTION
`auto`	start the AUTO2000 Command Line User Interface
AUTO -COMMAND	ACTION
`cd`	go to main directory (or other directory).
`! mkdir ab`	create an empty work directory. Note: the '!' is used to signify a command which is sent to the shell.
`cd ab`	change to the work directory.
`demo('ab')`	copy the demo files to the work directory.

Table 7.1: Copying the demo ab files.

At this point you may want to see what files have been copied to the work directory. In particular, you may want to edit the equations-file ab.c to see how the equations have been entered (in subroutine **func**) and how the starting solution has been set (in subroutine **stpnt**). Note that, initially, $p_1 = 0$ $p_2 = 14$, and $p_3 = 2$, for which $u_1 = u_2 = 0$ is a stationary solution.

7.4 Executing all Runs Automatically.

To execute all prepared runs of demo `ab`, simply type one or both of the command given in Table 7.2.

AUTO -COMMAND	ACTION
`demofile('ab_old.auto')`	execute all runs of demo `ab` interactively using a new constants file for each run
`demofile('ab_new.auto')`	execute all runs of demo `ab` interactively by modifying the constants file before each run

Table 7.2: Executing all runs of demo `ab`.

Each of the commands in Table 7.2 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 `ab_old.auto` performs the demo by reading in a sequence of AUTO constants files each of which corresponds to a step of the demo. The tutorial script `ab_new.auto` performs the demo by reading in a single AUTO constants file and then interactively modifying it to perform each of the demo. Both are valid and effective methods for running AUTO , with `ab_old.auto` being similar to the way AUTO was used before the advent of the CLUI, and `ab_new.auto` using new functionality provided by the CLUI.

Note that there are five separate runs. In the first run, a branch of stationary solutions is traced out. Along it, two folds (LP) and one Hopf bifurcation (HB) are located. The free parameter is p_1. The other parameters remain 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.ab`, `s.ab`, and `d.ab`.

The second run traces out the branch of periodic solutions that emanates from the Hopf bifurcation. The free parameters are p_1 and the period. The detailed results are appended to the existing data-files `b.ab`, `s.ab`, and `d.ab`.

In the third run, one of the folds detected in the first run is followed in the two parameters p_1 and p_3, while p_2 remains fixed. The fourth run continues this branch in opposite direction. Similarly, in the fifth run, the Hopf bifurcation located in the first run is followed in the two parameters p_1 and p_3. (In this example this is done in one direction only.) The detailed results of these continuations are accumulated in the data-files `b.2p`, `s.2p`, and `d.2p`.

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.

One could now use the AUTO CLUI to graphically inspect the contents of the data-files, but we shall do this later. However, it may be useful to edit these files to view their contents.

Next, reset the work directory, by typing the command given in Table 7.3.

AUTO -COMMAND	ACTION
`cl()`	remove temporary files of demo `ab`
`dl('ab')`	remove 'ab' data-files of demo `ab`
`dl('2p')`	remove '2p' data-files of demo `ab`

Table 7.3: Cleaning the demo `ab` work directory.

```
ab : first run : stationary solutions

BR    PT  TY LAB   PAR(1)       L2-NORM       U(1)         U(2)
 1     1  EP   1  0.00000E+00  0.00000E+00  0.00000E+00  0.00000E+00
 1    33  LP   2  1.05739E-01  1.48439E+00  3.11023E-01  1.45144E+00
 1    70  LP   3  8.89318E-02  3.28824E+00  6.88982E-01  3.21525E+00
 1    90  HB   4  1.30899E-01  4.27186E+00  8.95080E-01  4.17704E+00
 1    92  EP   5  1.51241E-01  4.36974E+00  9.15589E-01  4.27275E+00
 Saved as *.ab

ab : second run : periodic solutions

BR    PT  TY LAB   PAR(1)       L2-NORM      MAX U(1)     MAX U(2)     PERIOD
 4    30       6  1.19881E-01  3.98712E+00  9.91911E-01  7.02034E+00  2.721E+00
 4    60       7  1.15303E-01  3.14630E+00  9.99577E-01  9.95764E+00  6.147E+00
 4    90       8  1.05650E-01  2.21917E+00  9.99166E-01  9.36609E+00  1.399E+01
 4   120       9  1.05507E-01  1.69684E+00  9.99086E-01  9.29629E+00  9.956E+01
 4   150  EP  10  1.05507E-01  1.60388E+00  9.99789E-01  9.28146E+00  1.867E+03
 Appended to *.ab

ab : third run : a 2-parameter locus of folds

BR    PT  TY LAB   PAR(1)       L2-NORM       U(1)         U(2)        PAR(3)
 2    27  LP  11  1.35335E-01  2.06012E+00  4.99653E-01  1.99861E+00  2.499E+00
 2   100  EP  12  1.09381E-08  2.13650E+01  9.53147E-01  2.13437E+01 -3.748E-01
 Saved as *.2p

ab : fourth run : the locus of folds in reverse direction

BR    PT  TY LAB   PAR(1)       L2-NORM       U(1)         U(2)        PAR(3)
 2    35  EP  11 -1.31939E-03  9.96432E-01 -3.58651E-03  9.96426E-01 -1.050E+00
 Appended to *.2p

ab : fifth run : a 2-parameter locus of Hopf points

BR    PT  TY LAB   PAR(1)       L2-NORM       U(1)         U(2)        PAR(3)
 4   100  EP  11  8.80940E-05  1.17440E+01  9.14609E-01  1.17083E+01  9.362E-02
 Appended to *.2p
```

7.5 Executing Selected Runs Automatically.

As illustrated by the commands in Table 7.6, one can also execute selected runs of demo `ab`. In general, this cannot be done in arbitrary order, as any given run may need restart data from a previous run. Run 3 only requires the results of Run 1, so that the displayed command sequence is indeed appropriate. The screen output of these runs will be identical to that of the corresponding earlier runs, except for a change in solution labels in Run 3.

In real use there are two mains ways in which the AUTO can be used. First, one can prepare a constants-file for each run. In the illustrative runs below, the constants-files were carefully prepared in advance. For example, the file `c.ab.1` contains the AUTO -constants for Run 1, `c.ab.3` contains the AUTO -constants for Run 3, etc.

AUTO -COMMAND	ACTION
`ld("ab")`	load the problem definition `ab`
`run(c="ab.1")`	execute the run which uses the constants in `c.ab.1`
`sv("ab")`	save the results of the run into the files `b.ab`, `s.ab`, and `d.ab`
`run(c="ab.3",s="ab")`	execute the third run of demo `ab`

Table 7.4: Selected runs of demo `ab`.

On the other hand, one can use the CLUI to generate the constants file at runtime. In the example below, the constant file `c.ab.1` will be read in, and the CLUI will be used to make the appropriate changes to perform the same calculation as in Table 7.6.

AUTO -COMMAND	ACTION
`ld("ab")`	load the problem definition `ab`
`run(c="ab.1")`	execute the run which uses the constants in `c.ab.1`
`sv("ab")`	save the results of the run into the files `b.ab`, `s.ab`, and `d.ab`
`cc("IRS",2)`	start the new calculation from a solution with label 2
`cc("ICP",[0,2])`	since we are following a locus of folds we require two free parameters
`cc("ISP",0)`	turn off detection of branch points
`cc("ISW",2)`	since we start at a fold the ISW parameter indicates we desire to compute a locus of such points
`cc("DSMAX",0.5)`	increase the maximum allowed step size
`run(s="ab")`	execute the third run of demo `ab`

Table 7.5: Selected runs of demo `ab`.

7.6 Using AUTO -Commands.

Next, with the commands in Table ??, we execute the first two runs of demo `ab` again, using commands similar Table ?? that one would normally use in an actual application. We still use

AUTO -COMMAND	ACTION
`cl()`	remove temporary files of any previous runs of the demo
`dl("ab")`	remove 'ab' data-files of any previous runs of the demo
`dl("2p")`	remove '2p' data-files of any previous runs of the demo
`ld("ab")`	make sure the problem definition is loaded
`run(c="ab.1")`	compute a stationary solution branch with folds and Hopf bifurcation
`sv("ab")`	save output-files as b.ab, s.ab, d.ab
`run(c="ab.2",s="ab")`	compute a branch of periodic solutions from the Hopf point
`ap("ab")`	append the output-files to b.ab, s.ab, d.ab

Table 7.6: Commands for Run 1 and Run 2 of demo ab.

the demo constants-files that were prepared in advance and assume you are in the directory into which the ab demo has already been copied

It is instructive to look at the constants-files c.ab.1 and c.ab.2 used in the two runs above. The significance of each AUTO -constant set in these files can be found in Chapter 5. Note in particular the AUTO -constants that were changed between the two runs; see Table 7.7.

Constant	Run 1	Run 2	Reason for Change
IPS	1	2	To compute periodic solutions in Run 2
IRS	0	4	To specify the Hopf bifurcation restart label
NICP	1	2	The second run has two free parameters
ICP	1	1, 11	To use and print PAR(1) and PAR(11) in Run 2
NMX	100	150	To allow more continuation steps in Run 2
NPR	100	30	To print output every 30 steps in Run 2

Table 7.7: Differences in AUTO -constants between c.ab.1 and c.ab.2.

Actually, for periodic solutions, AUTO automatically adds PAR(11) (the period) as second parameter. However, for the period to be printed, one must specify the index 11 in the ICP list, as shown in Table 7.7.

7.7 Plotting the Results with AUTO .

The bifurcation diagram computed in the runs above is stored in the file b.ab, while each labeled solution is fully stored in s.ab. To use AUTO to graphically inspect these data-files, type the AUTO -command given in Table 7.8. The saved plots are shown in Figure 7.1 and in Figure 7.2.

Figure 7.1 shows the default view of the plotting tool, which consists of a representation of the bifurcation diagram. Step by step instructions for creating Figure 7.2 are given below.

The plotting window consists of a menubar at the top, a plotting area, and a control panel with four control widgets at the bottom. The first step in creating Figure 7.2 is to 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 s.ab. Unfortunately, this is an equilibrium solution, so only a single point is plotted. Since we wish to plot the periodic solutions, we modify the widget marked "Label" by changing its value from "[1]" to "[6,7,10]" (don't forget to hit the return key when you are done modifying the value). This signifies that instead of plotting the solution with label 1 we want to plot the solutions with labels 6, 7, and 10 simultaneously. In the plotting window we now have three curves, each of which is a plot of time versus the value of the first state variable. If we want a different plot, say the values of the two state variables plotted against each other, we use the two remaining widgets in the control panel, labeled "X" and "Y". For example, if change the value of "X" from "['t']" to "[0]" and the value of "Y" from "[0]" to "[1]" we get a phase plot of the period solutions (don't forget to hit the return key when you are done modifying each value). This plot is shown in Figure 7.2.

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 4.10.

AUTO -COMMAND	ACTION
`plot("ab")`	run AUTO to graph the contents of b.ab and s.ab;

Table 7.8: Command for plotting the files b.ab and s.ab.

7.8 Following Folds and Hopf Bifurcations.

The commands in Table 7.9 will execute the remaining runs of demo ab. Here, as in later demos, some of the AUTO -constants that have been changed between runs are indicated in the Table.

AUTO -COMMAND	ACTION
`run(c="ab.3",s="ab")`	compute a locus of folds with changes (from c.ab.1) : IRS, NICP, ICP, ISW, DSMAX
`sv("2p")`	save output-files as b.2p, s.2p, d.2p
`run(c="ab.4",s="ab")`	compute the locus of folds in reverse direction with changes (from c.ab.3) : DS (sign)
`ap("2p")`	append the output-files to b.2p, s.2p, d.2p
`run(c="ab.4",s="ab")`	compute a locus of Hopf points with changes (from c.ab.4) : IRS
`ap("2p")`	append the output-files to b.2p, s.2p, d.2p

Table 7.9: Commands for Runs 3, 4, and 5 of demo ab.

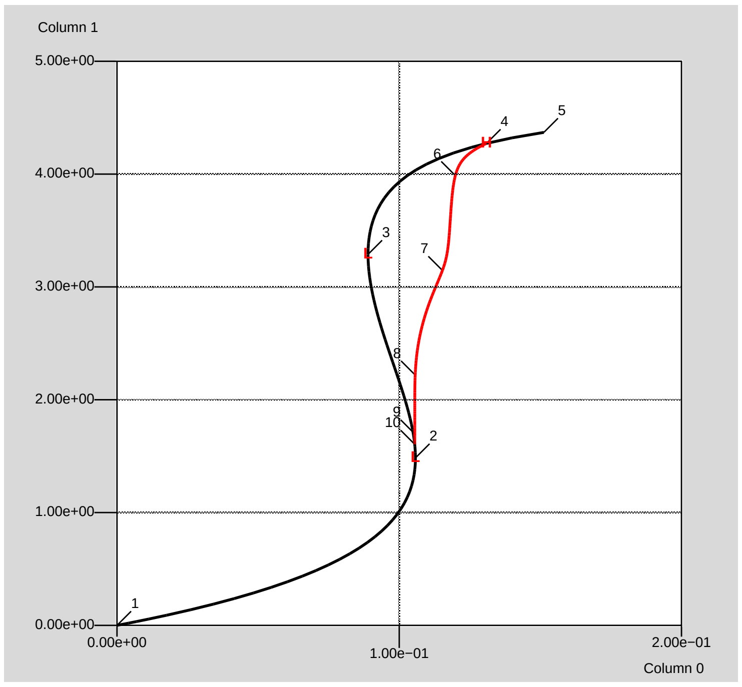


Figure 7.1: The bifurcation diagram of demo ab.

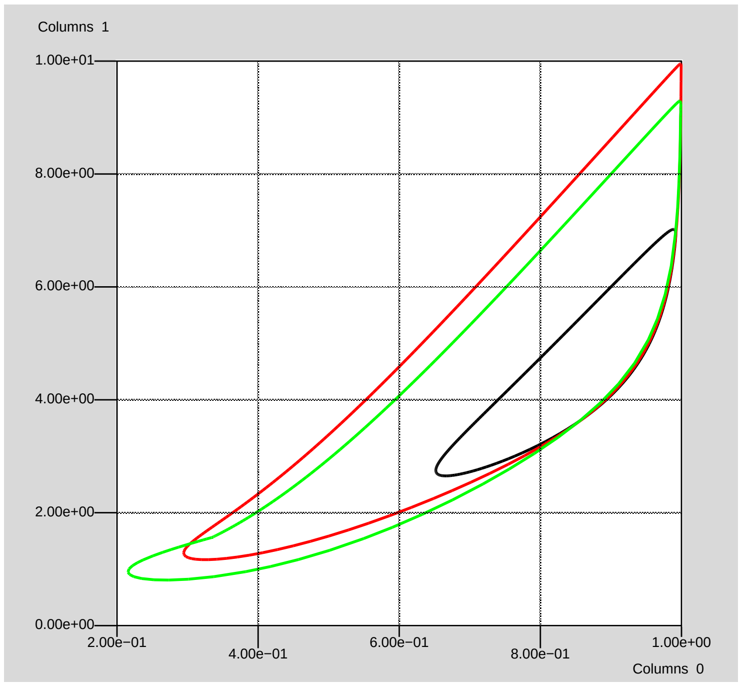


Figure 7.2: The phase plot of solutions 6, 7, and 10 in demo ab.

7.9 Relabeling Solutions in the Data-Files.

Next we want to plot the two-parameter diagram computed in the last three runs. However, the solution labels in these runs are not distinct. This is due to the fact that in each of these three runs the restart solution was read from `s.ab`, while the computed solutions were stored in `s.2p`. Consequently, these runs were unaware of each other's results, which led to non-unique labels. For relabeling purpose, and more generally for file maintenance, there is a utility program that can be invoked as indicated in Table 7.10. Its use is illustrated in Table 7.11.

AUTO -COMMAND	ACTION
`rl("2p")`	run the relabeling program on `b.2p` and `s.2p`

Table 7.10: Command to run the relabeling program on `b.2p` and `s.2p`.

RELABELING COMMAND	ACTION
l	list the labeled solutions in `s.2p`
r	relabel the solutions
l	list the new solution labeling
w	rewrite `b.2p` and `s.2p`

Table 7.11: Relabeling commands for the files `b.2p` and `s.2p`.

7.10 Plotting the 2-Parameter Diagram.

To plot the files `b.2p` and `s.2p`, enter the command listed in Table 7.12. The saved plot is shown in Figure 7.3.

AUTO -COMMAND	ACTION
`plot("2p")`	run to graph the contents of `b.2p` and `s.2p`;

Table 7.12: Command to plot the files `b.2p` and `s.2p`.

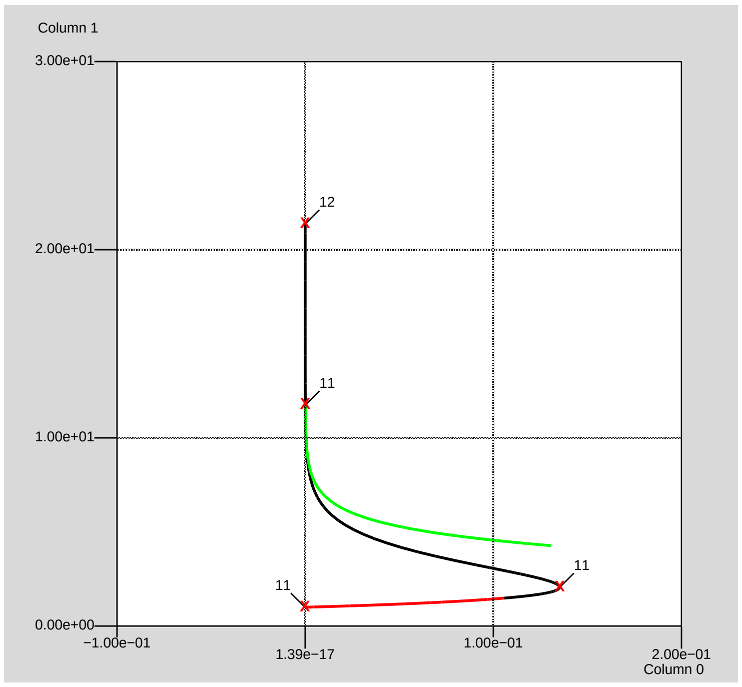


Figure 7.3: Loci of folds and Hopf bifurcations for demo `ab`.

Chapter 8

AUTO Demos : Fixed points.

8.1 enz : Stationary Solutions of an Enzyme Model.

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

$$\begin{aligned} s_1' &= (s_0 - s_1) + (s_2 - s_1) - \rho R(s_1), \\ s_2' &= (s_0 + \mu - s_2) + (s_1 - s_2) - \rho R(s_2), \end{aligned} \tag{8.1}$$

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 (1991*a*).

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

Table 8.1: Commands for running demo enz.

8.2 dd2 : Fixed Points of a Discrete Dynamical System.

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

$$\begin{aligned} u_1^{k+1} &= p_1 u_1^k(1-u_1^k) - p_2 u_1^k u_2^k, \\ u_2^{k+1} &= (1-p_3)u_2^k + p_2 u_1^k u_2^k. \end{aligned} \tag{8.2}$$

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

Table 8.2: Commands for running demo dd2.

Chapter 9

AUTO Demos : Periodic solutions.

9.1 lrz : The Lorenz Equations.

This demo computes two symmetric homoclinic orbits in the Lorenz equations

$$\begin{aligned} u_1' &= p_3(u_2 - u_1), \\ u_2' &= p_1u_1 - u_2 - u_1u_3, \\ u_3' &= u_1u_2 - p_2u_3. \end{aligned} \tag{9.1}$$

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 branches.

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

Table 9.1: Commands for running demo lrz.

9.2 abc : The A → B → C Reaction.

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

$$
\begin{aligned}
u_1' &= -u_1 + p_1(1-u_1)e^{u_3},\\
u_2' &= -u_2 + p_1 e^{u_3}(1-u_1-p_5u_2),\\
u_3' &= -u_3 - p_3u_3 + p_1p_4e^{u_3}(1-u_1+p_2p_5u_2),
\end{aligned}
\tag{9.2}
$$

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

AUTO -COMMAND	ACTION
`! mkdir abc`	create an empty work directory
`cd abc`	change directory
`demo('abc')`	copy the demo files to the work directory
`ld('abc')`	load the problem definition
`run(c='abc.1')`	compute the stationary solution branch with Hopf bifurcations
`sv('abc')`	save output-files as b.abc, s.abc, d.abc
`run(c='abc.2',s='abc')`	compute a branch of periodic solutions from the first Hopf point. Constants changed : `IRS, IPS, NICP, ICP`
`ap('abc')`	append the output-files to b.abc, s.abc, d.abc
`run(c='abc.3',s='abc')`	compute a branch of periodic solutions from the second Hopf point. Constants changed : `IRS, NMX`
`ap('abc')`	append the output-files to b.abc, s.abc, d.abc

Table 9.2: Commands for running demo abc.

9.3 pp2 : A 2D Predator-Prey Model.

This demo illustrates a variety of calculations. The equations, which model a predator-prey system with harvesting, are

$$\begin{aligned} u_1' &= p_2 u_1(1-u_1) - u_1 u_2 - p_1(1-e^{-p_3 u_1}), \\ u_2' &= -u_2 + p_4 u_1 u_2. \end{aligned} \tag{9.3}$$

Here p_1 is the principal continuation parameter, $p_3 = 5$, $p_4 = 3$, and, initially, $p_2 = 3$. For two-parameter computations p_2 is also free.

AUTO -COMMAND	ACTION
`! mkdir pp2`	create an empty work directory
`cd pp2`	change directory
`demo('pp2')`	copy the demo files to the work directory
`ld('pp2')`	load the problem definition
`run(c='pp2.1')`	1st run; stationary solutions
`sv('pp2')`	save output-files as b.pp2, s.pp2, d.pp2
`run(c='pp2.2',s='pp2')`	2nd run; restart at a labeled solution. Constants changed : `IRS, RL1`
`ap('pp2')`	append output-files to b.pp2, s.pp2, d.pp2
`run(c='pp2.3',s='pp2')`	3rd run; periodic solutions. Constants changed : `IRS, IPS, ILP`
`ap('pp2')`	append output-files to b.pp2, s.pp2, d.pp2
`run(c='pp2.4',s='pp2')`	4th run; restart at a labeled periodic solution. Constants changed : `IRS, NTST`
`ap('pp2')`	append output-files to b.pp2, s.pp2, d.pp2
`run(c='pp2.5',s='pp2')`	5th run; continuation of folds. Constants changed : `IRS, IPS, ISW, ICP`
`sv('lp')`	save output-files as b.lp, s.lp, d.lp
`run(c='pp2.6',s='pp2')`	6th run; continuation of Hopf bifurcations. Constants changed : `IRS`
`sv('hb')`	save output-files as b.hb, s.hb, d.hb
`run(c='pp2.7',s='pp2')`	7th run; continuation of homoclinic orbits. Constants changed : `IRS, IPS, ISP`
`sv('hom')`	save output-files as b.hom, s.hom, d.hom

Table 9.3: Commands for running demo pp2.

9.4 lor : Starting an Orbit from Numerical Data.

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

$$\begin{aligned} u_1' &= p_3(u_2 - u_1), \\ u_2' &= p_1 u_1 - u_2 - u_1 u_3, \\ u_3' &= u_1 u_2 - p_2 u_3. \end{aligned} \tag{9.4}$$

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

Table 9.4: Commands for running demo lor.

9.5 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

$$\begin{aligned} x' &= x + \beta y - x(x^2+y^2), \\ y' &= -\beta x + y - y(x^2+y^2), \end{aligned} \tag{9.5}$$

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

$$\begin{aligned} v' &= \big(F(v) - w\big)/\epsilon, \\ w' &= v - dw - \big(b + r\sin(\beta t)\big), \end{aligned} \tag{9.6}$$

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 branch with $r = 0.2$ and varying β is computed in the second run. For detailed results see Alexander, Doedel & Othmer (1990).

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

Table 9.5: Commands for running demo frc.

9.6 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

$$\begin{aligned} u_1' &= u_1(1-u_1) - p_4 u_1 u_2, \\ u_2' &= -p_2 u_2 + p_4 u_1 u_2 - p_5 u_2 u_3 - p_1(1 - e^{-p_6 u_2}) \\ u_3' &= -p_3 u_3 + p_5 u_2 u_3. \end{aligned} \tag{9.7}$$

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

Table 9.6: Commands for running demo ppp.

9.7 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 branch 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

$$\begin{aligned} s' &= (s_0 - s) - \rho R(s,a), \\ a' &= \alpha(a_0 - a) - \rho R(s,a), \end{aligned} \tag{9.8}$$

where

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

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

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

Table 9.7: Commands for running demo plp.

9.8 pp3 : Period-Doubling Continuation.

This demo illustrates the computation of stationary solutions, Hopf bifurcations, and periodic solutions, branch switching at a period-doubling bifurcation, and the computation of a locus of period-doubling bifurcations. The equations model a 3D predator-prey system with harvesting (Doedel (1984)).

$$\begin{aligned} u_1' &= u_1(1-u_1) - p_4 u_1 u_2, \\ u_2' &= -p_2 u_2 + p_4 u_1 u_2 - p_5 u_2 u_3 - p_1(1-e^{-p_6 u_2}) \\ u_3' &= -p_3 u_3 + p_5 u_2 u_3. \end{aligned} \tag{9.9}$$

The free parameter is p_1, except in the period-doubling continuation, where both p_1 and p_2 are free.

AUTO -COMMAND	ACTION
`! mkdir pp3`	create an empty work directory
`cd pp3`	change directory
`demo('pp3')`	copy the demo files to the work directory
`ld('pp3')`	load the problem definition
`run(c='pp3.1')`	1st run; stationary solutions
`sv('pp3')`	save output-files as b.pp3, s.pp3, d.pp3
`run(c='pp3.2',s='pp3')`	compute a branch of periodic solutions. Constants changed : `IRS, IPS, NMX`
`ap('pp3`	append output-files to b.pp3, s.pp3, d.pp3
`run(c='pp3.3',s='pp3')`	compute the branch bifurcating at the period-doubling. Constants changed : `IRS, ISW, NTST`
`ap('pp3')`	append output-files to b.pp3, s.pp3, d.pp3
`run(c='pp3.4',s='pp3')`	generate starting data for the period-doubling continuation. Constants changed : `ISW`
`sv('tmp')`	save output-files as b.tmp, s.tmp, d.tmp
`run(c='pp3.5',s='tmp')`	period-doubling continuation; restart from s.tmp. Constants changed : `IRS`
`sv('2p')`	save output-files as b.2p, s.2p, d.2p

Table 9.8: Commands for running demo pp3.

9.9 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

$$\begin{aligned} x'(t) &= \big[-(\beta+\nu)x + \beta y - a_3 x^3 + b_3 (y-x)^3\big]/r, \\ y'(t) &= \beta x - (\beta+\gamma) y - z - b_3 (y-x)^3, \\ z'(t) &= y, \end{aligned} \tag{9.10}$$

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

Table 9.9: Commands for running demo tor.

9.10 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

$$\begin{aligned} \phi_1'' + \epsilon\phi_1' + \sin\phi_1 &= I + \gamma(\phi_2 - \phi_1), \\ \phi_2'' + \epsilon\phi_2' + \sin\phi_2 &= I + \gamma(\phi_1 - \phi_2), \end{aligned} \tag{9.11}$$

or, in equivalent first order form,

$$\begin{aligned} \phi_1' &= \psi_1, \\ \phi_2' &= \psi_2, \\ \psi_1' &= -\epsilon\psi_1 - \sin\phi_1 + I + \gamma(\phi_2 - \phi_1), \\ \psi_2' &= -\epsilon\psi_2 - \sin\phi_2 + I + \gamma(\phi_1 - \phi_2). \end{aligned} \tag{9.12}$$

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 correponding 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 branch is computed in the second run. Two branch points are located, and the bifurcating branches 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 branch of period-doubled (and out-of-phase) rotations. Constants changed : `IPS, NTST, ISW, NMX`
	append output-files tp b.pen, s.pen, d.pen

`run(c='pen.3',s='pen')`
 	
`ap('pen')`
	a secondary bifurcating branch (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 branch (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 9.10: Commands for running demo pen.

9.11 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

$$\begin{aligned} u_1' &= \alpha \left[\, u_2 - h(u_1) \, \right] \, , \\ u_2' &= u_1 - u_2 + u_3 \, , \\ u_3' &= -\beta \, u_2 \, , \end{aligned} \tag{9.13}$$

where

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

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

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

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

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

Table 9.11: Commands for running demo chu.

9.12 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

$$\begin{aligned} u_1' &= \lambda u_1 - u_2, \\ u_2' &= u_1(1-u_1). \end{aligned} \tag{9.14}$$

This equation has a Hopf bifurcation from the trivial solution at $\lambda = 0$. The bifurcating branch of periodic solutions is vertical and along it the period increases monotonically. The branch 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
`! mkdir phs`	create an empty work directory
`cd phs`	change directory
`demo('phs')`	copy the demo files to the work directory
`ld('phs')`	load the problem definition
`run(c='phs.1')`	detect Hopf bifurcation
`sv('phs')`	save output-files as b.phs, s.phs, d.phs
`run(c='phs.2',s='phs')`	compute periodic solutions. Constants changed : `IRS, IPS, NPR`
`ap('phs')`	append output-files to b.phs, s.phs, d.phs

Table 9.12: Commands for running demo phs.

9.13 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 :

$$\begin{aligned} u_1' &= p_2 u_1(1-u_1) - u_1 u_2 - p_1(1-e^{-p_3 u_1}), \\ u_2' &= -u_2 + p_4 u_1 u_2, \end{aligned} \tag{9.15}$$

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

Table 9.13: Commands for running demo ivp.

Chapter 10

AUTO Demos : BVP.

10.1 exp : Bratu's Equation.

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

$$\begin{aligned} u_1' &= u_2, \\ u_2' &= -p_1 e^{u_1}, \end{aligned} \tag{10.1}$$

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

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

Table 10.1: Commands for running demo exp.

10.2 int : Boundary and Integral Constraints.

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

$$\begin{aligned} u_1' &= u_2, \\ u_2' &= -p_1 e^{u_1}, \end{aligned} \tag{10.2}$$

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

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

The solution branch 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 10.2: Commands for running demo int.

10.3 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 branch. The branch of solutions that bifurcates at the first eigenvalue is computed in both directions. The equations are

$$\begin{aligned} u_1' &= u_2, \\ u_2' &= -(p_1\pi)^2 u_1 + u_1^2, \end{aligned} \tag{10.3}$$

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

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

Table 10.3: Commands for running demo bvp.

10.4 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 branch. 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

$$\begin{aligned} u_1' &= u_2, \\ u_2' &= (p_1\pi)^2 u_1, \end{aligned} \tag{10.4}$$

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

Table 10.4: Commands for running demo lin.

10.5 non : A Non-Autonomous BVP.

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

$$\begin{aligned} u_1' &= u_2, \\ u_2' &= -p_1 e^{x^3 u_1}, \end{aligned} \tag{10.5}$$

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

$$\begin{aligned} u_1' &= u_2, \\ u_2' &= -p_1 e^{u_3^3 u_1}, \\ u_3' &= 1, \end{aligned} \tag{10.6}$$

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

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

Table 10.5: Commands for running demo non.

10.6 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) :

$$\begin{aligned} u_1' &= T u_2, \\ u_2' &= T u_3, \\ u_3' &= T\big[-2\gamma u_4 + u_2^2 - 2u_1u_3 - u_4^2\big], \\ u_4' &= T u_5, \\ u_5' &= T\big[2\gamma u_2 + 2u_2u_4 - 2u_1u_5\big], \end{aligned} \tag{10.7}$$

with left boundary conditions

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

and (asymptotic) right boundary conditions

$$\begin{aligned} &\big[f_\infty + a(f_\infty,\gamma)\big]\ u_2(1) + u_3(1) - \gamma\ \tfrac{u_4(1)}{a(f_\infty,\gamma)} = 0, \\ &a(f_\infty,\gamma)\ \tfrac{b^2(f_\infty,\gamma)}{\gamma}\ u_2(1) + \big[f_\infty + a(f_\infty,\gamma)\big]\ u_4(1) + u_5(1) = 0, \\ &u_1(1) = f_\infty, \end{aligned} \tag{10.8}$$

where

$$\begin{aligned} a(f_\infty,\gamma) &= \tfrac{1}{\sqrt{2}}\big[(f_\infty^4 + 4\gamma^2)^{1/2} + f_\infty^2\big]^{1/2}, \\ b(f_\infty,\gamma) &= \tfrac{1}{\sqrt{2}}\big[(f_\infty^4 + 4\gamma^2)^{1/2} - f_\infty^2\big]^{1/2}. \end{aligned} \tag{10.9}$$

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

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

Table 10.6: Commands for running demo kar.

10.7 spb : A Singularly-Perturbed BVP.

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

$$\begin{aligned} u_1' &= u_2, \\ u_2' &= \tfrac{\lambda}{\epsilon}\big(u_1 u_2 (u_1^2 - 1) + u_1\big), \end{aligned} \tag{10.10}$$

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

Table 10.7: Commands for running demo spb.

10.8 ezp : Complex Bifurcation in a BVP.

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

$$\begin{aligned} u_1' &= u_2, \\ u_2' &= -p_1 e^{u_1}, \end{aligned} \tag{10.11}$$

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 branch 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 branch are computed. On it, both solution components u_1 and u_2 have nontrivial imaginary part.

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

Table 10.8: Commands for running demo ezp.

Chapter 11

AUTO Demos : Parabolic PDEs.

11.1 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 $L =$ `PAR(11)` $= 10$ is fixed throughout, as is the diffusion constant $D =$ `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 `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 `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.c`. 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
`!  mkdir pd1`	create an empty work directory
`cd pd1`	change directory
`demo('pd1')`	copy the demo files to the work directory
`run(c='pd1.1')`	time integration towards stationary state
`sv('1')`	save output-files as `b.1`, `s.1`, `d.1`
`run(c='pd1.2',s='1')`	continuation of stationary states; read restart data from `s.1`. constants changed : `IPS, IRS, ICP`, etc.
`sv('2')`	save output-files as `b.2`, `s.2`, `d.2`

Table 11.1: Commands for running demo `pd1`.

11.2 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

$$\begin{aligned} \partial u_1/\partial t &= D_1\, \partial^2 u_1/\partial x^2 \;+\; p_1\, u\,(1-u) \;-\; u_1 u_2, \\ \partial u_2/\partial t &= D_2\, \partial^2 u_2/\partial x^2 \;-\; u_2 \;+\; u_1 u_2, \end{aligned} \tag{11.1}$$

on the space interval $[0, L]$, where $L =$ `PAR(11)` $= 1$ is fixed throughout, as are the diffusion constants $D_1 =$ `PAR(15)` $= 1$ and $D_2 =$ `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 `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 `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.c. 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
`! mkdir pd2`	create an empty work directory
`cd pd2`	change directory
`demo('pd2')`	copy the demo files to the work directory
`run(c='pd2.1')`	time integration towards stationary state
`sv('1')`	save output-files as b.1, s.1, d.1
`run(c='pd2.2',s='1')`	continuation of stationary states; read restart data from s.1. constants changed : `IPS, IRS, ICP`, etc.
`sv('2')`	save output-files as b.2, s.2, d.2

Table 11.2: Commands for running demo pd2.

11.3 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 (1986*b*)) are :

$$\begin{aligned} \partial u_1/\partial t &= \partial^2 u_1/\partial x^2 - p_1\big[p_4 R(u_1,u_2) - (p_2 - u_1)\big], \\ \partial u_2/\partial t &= \beta\partial^2 u_2/\partial x^2 - p_1\big[p_4 R(u_1,u_2) - p_7(p_3 - u_2)\big]. \end{aligned} \tag{11.2}$$

All equation parameters, except p_3, are fixed throughout.

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

Table 11.3: Commands for running demo wav.

11.4 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))

$$\begin{aligned} u_t &= D_x/L^2 u_{xx} + u^2 v - (B+1)u + A, \\ v_t &= D_y/L^2 v_{xx} - u^2 v + Bu, \end{aligned} \tag{11.3}$$

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

Table 11.4: Commands for running demo brc.

11.5 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))

$$\begin{aligned} u_t &= D_x/L^2 u_{xx} + u^2 v - (B+1)u + A, \\ v_t &= D_y/L^2 v_{xx} - u^2 v + Bu, \end{aligned} \tag{11.4}$$

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

Table 11.5: Commands for running demo brf.

11.6 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

$$\begin{aligned} u_t &= D_x/L^2 u_{xx} + u^2 v - (B+1)u + A, \\ v_t &= D_y/L^2 v_{xx} - u^2 v + Bu, \end{aligned} \tag{11.5}$$

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.c.

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

Table 11.6: Commands for running demo bru.

Chapter 12

AUTO Demos : Optimization.

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

Table 12.1: Commands for running demo opt.

12.2 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

$$\begin{aligned} x'(t) &= [-\lambda_4(x^3/3 - x) + (z - x)/\lambda_2 - y]/\lambda_1, \\ y'(t) &= x - \lambda_3, \\ z'(t) &= -(z - x)/\lambda_2, \end{aligned} \tag{12.1}$$

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.c. For reference purpose it is convenient here to write down the full extended system in its general form :

$$\begin{aligned}
&u'(t) = Tf\big(u(t), \lambda\big), \qquad T \in \mathrm{R}\ \text{(period)},\ u(\cdot), f(\cdot,\cdot) \in \mathrm{R}^n,\ \lambda \in \mathrm{R}^{n_\lambda}, \\
&w'(t) = -Tf_u\big(u(t), \lambda\big)^* w(t) + \kappa u_0'(t) + \gamma g_u\big(u(t), \lambda\big)^*, \qquad w(\cdot) \in \mathrm{R}^n,\ \kappa, \gamma \in \mathrm{R}, \\
&u(1) - u(0) = 0, \qquad w(1) - w(0) = 0, \\
&\textstyle\int_0^1 u(t)^* u_0'(t)\ dt = 0, \\
&\textstyle\int_0^1 \omega - g\big(u(t), \lambda\big)\ dt = 0, \\
&\textstyle\int_0^1 w(t)^* w(t) + \kappa^2 + \gamma^2 - \alpha\ dt = 0, \qquad \alpha \in \mathrm{R}, \\
&\textstyle\int_0^1 f\big(u(t), \lambda\big)^* w(t) - \gamma g_T\big(u(t), \lambda\big) - \tau_0\ dt = 0, \qquad \tau_0 \in \mathrm{R}, \\
&\textstyle\int_0^1 Tf_{\lambda_i}\big(u(t), \lambda\big)^* w(t) - \gamma g_{\lambda_i}\big(u(t), \lambda\big) - \tau_i\ dt = 0, \qquad \tau_i \in \mathrm{R}, \quad i = 1, \cdots, n_\lambda.
\end{aligned} \tag{12.2}$$

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

In the computations below, the two preliminary runs, with `IPS`=1 and `IPS`=2, respectively, locate periodic solutions. The subsequent runs are with `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 branch of periodic solutions from the Hopf bifurcation.
- `Run 3.` This run retraces part of the periodic solution branch, 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 branch 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 branch.
- `Run 5.` The above-found starting solution is continued in two system parameters, here λ_3 and λ_2; i.e., a two-parameter branch of extrema with respect to λ_3 is computed. Along this branch 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 branch of transcritical secondary periodic bifurcations points.

Although not illustrated here, one can restart an ordinary continuation of periodic solutions, using `IPS`=2 or `IPS`=3, from a labeled solution point on a branch computed with `IPS`=15.

The free scalar variables specified in the AUTO constants-files for Run 3 and Run 4 are shown in Table 12.2.

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

Table 12.2: `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 12.3: `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 12.4: `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 branch 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 bifurcation; restart from s.0. Constants changed : `IPS, IRS, ICP`, $\cdots$
	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 branch of 1-parameter extrema; restart from s.1. Constants changed : `IRS, ISW, ICP, ISW`, $\cdots$
	save the output-files as b.2, s.2, d.2

`run(c='ops.6',s='2')`
 	
`sv('3')`
	compute 3-parameter branch 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 12.5: Commands for running demo ops.

12.3 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 subroutines in `obv.c`. For the case of periodic solutions the optimality system can be generated automatically; see the demo `ops`.

Consider the system

$$\begin{aligned} u_1'(t) &= u_2(t), \\ u_2'(t) &= -\lambda_1 e^{p(u_1,\lambda_2,\lambda_3)}, \end{aligned} \tag{12.3}$$

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

$$\begin{aligned} u_1(0) &= 0, \\ u_1(1) &= 0. \end{aligned} \tag{12.4}$$

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

$$\begin{aligned} u_1'(t) &= u_2(t), \\ u_2'(t) &= -\lambda_1 e^{p(u_1,\lambda_2,\lambda_3)}, \\ w_1'(t) &= \lambda_1 e^{p(u_1,\lambda_2,\lambda_3)} p_{u_1} w_2(t) + 2\gamma(u_1(t)-1), \\ w_2'(t) &= -w_1(t), \end{aligned} \tag{12.5}$$

where

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

with

$$\begin{array}{lll} u_1(0)=0, & w_1(0)-\beta_1=0, & w_2(0)=0, \\ u_1(1)=0, & w_1(1)+\beta_2=0, & w_2(1)=0, \end{array} \tag{12.6}$$

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

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

$$\begin{aligned} &\textstyle\int_0^1 \left[-e^{p(u_1,\lambda_2,\lambda_3)} w_2(t) - \frac{1}{5}\gamma\lambda_1\right] dt = 0, \\ &\textstyle\int_0^1 \left[-\lambda_1 e^{p(u_1,\lambda_2,\lambda_3)} u_1(t)^2 w_2(t) - \frac{1}{5}\gamma\lambda_2 - \tau_2\right] dt = 0, \\ &\textstyle\int_0^1 \left[-\lambda_1 e^{p(u_1,\lambda_2,\lambda_3)} u_1(t)^4 w_2(t) - \frac{1}{5}\gamma\lambda_3 - \tau_3\right] dt = 0. \end{aligned} \tag{12.7}$$

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

Table 12.6: Commands for running demo obv.

Chapter 13

AUTO Demos : Connecting orbits.

13.1 fsh : A Saddle-Node Connection.

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

$$\begin{aligned} &w_t = w_{xx} + f(w), \qquad -\infty < x < \infty, \quad t > 0, \\ &f(w) \equiv w(1-w). \end{aligned} \tag{13.1}$$

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

$$\begin{aligned} u_1'(z) &= u_2(z), \\ u_2'(z) &= cu_2(z) - f\big(u_1(z)\big). \end{aligned} \tag{13.2}$$

Its fixed point $(0,0)$ has two positive eigenvalues when $c > 2$. The other fixed point, $(1,0)$, is a saddle point. A branch 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
`! mkdir fsh`	create an empty work directory
`cd fsh`	change directory
`demo('fsh')`	copy the demo files to the work directory
`run(c='fsh.1')`	continuation in the period T, with c fixed; no phase condition
`sv('0')`	save output-files as b.0, s.0, d.0
`run(c='fsh.2',s='0')`	continuation in c and T, with active phase condition. Constants changed : `IRS, ICP, NINT, DS`
`sv('fsh')`	save output-files as b.fsh, s.fsh, d.fsh

Table 13.1: Commands for running demo fsh.

13.2 nag : A Saddle-Saddle Connection.

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

$$\begin{aligned} &w_t = w_{xx} + f(w,a), \qquad -\infty < x < \infty, \quad t > 0, \\ &f(w,a) \equiv w(1-w)(w-a), \qquad 0 < a < 1. \end{aligned} \tag{13.3}$$

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

$$\begin{aligned} u_1'(z) &= u_2(z), \\ u_2'(z) &= cu_2(z) - f\big(u_1(z), a\big), \end{aligned} \tag{13.4}$$

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

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

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 subroutine **stpnt** in `nag.c` 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')`	compute part of first branch of heteroclinic orbits
`sv('nag')`	save output-files as `b.nag`, `s.nag`, `d.nag`
`run(c='nag.2',s='nag')`	compute first branch in opposite direction. Constants changed : `DS`
`ap('nag')`	append output-files to `b.nag`, `s.nag`, `d.nag`

Table 13.2: Commands for running demo `nag`.

13.3 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 $A(w) = a_1 w + a_2 w^2$, $B(w) = b_0 + b_1 w + b_2 w^2$, and $C(w) = c_0 + c_1 w + c_2 w^2$. 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 $u(z) \to \textit{constant}$ as $z \to -\infty$. 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

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

to a corresponding sharp traveling wave of

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

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 13.3.

	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 13.3: Problem coefficients in demo `stw`.

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

$$\begin{aligned} u_1'(z) &= u_2, \\ u_2'(z) &= \left[cu_2 - B(u_1)u_2^2 - C(u_1)\right]/A(u_1). \end{aligned} \tag{13.5}$$

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

$$\begin{aligned} u_1'(\tilde{z}) &= A(u_1)u_2, \\ u_2'(\tilde{z}) &= cu_2 - B(u_1)u_2^2 - C(u_1). \end{aligned} \tag{13.6}$$

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

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

$$\begin{aligned} u_1'(\xi) &= TA(u_1)u_2, \\ u_2'(\xi) &= T\big[cu_2 - B(u_1)u_2^2 - C(u_1)\big]. \end{aligned} \tag{13.7}$$

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

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

This solution has wave speed $c = 1$. In the limit as $T \to \infty$ 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 continously from the values for Case 1 to the values for Case 2.

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

Table 13.4: Commands for running demo stw.

Chapter 14

AUTO Demos : Miscellaneous.

14.1 pvl : Use of the Subroutine pvls.

Consider Bratu's equation

$$\begin{aligned} u_1' &= u_2, \\ u_2' &= -p_1 e^{u_1}, \end{aligned} \tag{14.1}$$

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 pvls.c, 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
`! mkdir pvl`	create an empty work directory
`cd pvl`	change directory
`demo('pvl')`	copy the demo files to the work directory
`run(c='pvl.1')`	compute a solution branch
`sv('pvl')`	save output-files as b.pvl, s.pvl, d.pvl

Table 14.1: Commands for running demo pvl.

14.2 ext : Spurious Solutions to BVB.

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

$$\begin{aligned} & u_1' - u_2 = 0, \\ & u_2' + \lambda^2\pi^2 \sin(u_1 + u_1^2 + u_1^3) = 0, \qquad t \in [0,1], \\ & u_1(0) = 0, \quad u_1(1) = 0. \end{aligned} \tag{14.2}$$

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 5.3. This example is also considered in Beyn & Doedel (1981) and Doedel, Keller & Kernévez (1991*b*).

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

Table 14.2: Commands for running demo ext.

14.3 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

$$\begin{aligned} u_i' &= u_i, \\ v_i' &= -p_1 \, e(u_i), \end{aligned} \tag{14.3}$$

$i = 1, \cdots,$ `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
`! mkdir tim`	create an empty work directory
`cd tim`	change directory
`demo('tim')`	copy the demo files to the work directory
`run(c='tim.1')`	Timing run
`sv('tim')`	save output-files as b.tim, s.tim, d.tim

Table 14.3: Commands for running demo tim.

Chapter 15

HomCont.

15.1 Introduction.

HomCont is a collection of subroutines 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 subroutines are in the file auto/2000/src/autlib5.c. 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.

15.2 HomCont Files and Subroutines.

In order to run HomCont one must prepare an equations file xxx.c, 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.c serves as a sample for new equation files. It contains the C subroutines **func**, **stpnt**, **pvls**, **bcnd**, **icnd** and **fopt**. The final three are dummy subroutines 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.

15.3 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.

15.3.1 NUNSTAB

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

15.3.2 NSTAB

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

15.3.3 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).

15.3.4 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 15.7 for more details.

15.3.5 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`. 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 15.7 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,` $N = 1, 2, 3, \ldots$: Homoclinic branch switching: this description is for reference only and we refer to Chapter 22 to see how this can be used in actual practice and to Oldeman, Champneys & B. (2001) for theory and background.

  The orbit is split into $N + 1$ parts and AUTO sees it as an $(N + 1)\times$`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...NPARX-1]`.

  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...NPARX-NDIM-1]` 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[19+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 =`par[9]`, T_N =`par[10]` and T_i =`par[18+2*i]` for $i = 1 \ldots N - 1$. 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 ε=`par[21]` 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 in 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 adhoint.

15.3.6 `NREV, IREV`

If `NREV=1` then it is assumed that the system is reversible under the transformation $t \to -t$ and $U(i) \to -U(i)$ 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`.

15.3.7 `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`.

15.3.8 `NPSI, IPSI`

Number and labels of activated test functions for detecting homoclinic bifurcations, see Section 15.6 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.

15.4 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 15.6 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`.

15.5 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 15.7 below).
- `PAR(21)-PAR(36)` : These parameters are used for storing the test functions (see Section 15.6).

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 15.6. Finally, the values of the `NPSI` activated test functions are written.

15.6 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 $i = 1$); $\mathrm{Re}(\mu_1) = -\mathrm{Re}(\lambda_1)$.
- $i = 5$: Neutrally-divergent saddle-focus (stable eigenvalues complex); $\mathrm{Re}(\lambda_1) = -\mathrm{Re}(\mu_1) - \mathrm{Re}(\mu_2)$.
- $i = 6$: Neutrally-divergent saddle-focus (unstable eigenvalues complex); $\mathrm{Re}(\mu_1) = -\mathrm{Re}(\lambda_1) - \mathrm{Re}(\lambda_2)$.
- $i = 7$: Three leading eigenvalues (stable); $\mathrm{Re}(\lambda_1) = -\mathrm{Re}(\mu_1) - \mathrm{Re}(\mu_2)$.
- $i = 8$: Three leading eigenvalues (unstable); $\mathrm{Re}(\mu_1) = -\mathrm{Re}(\lambda_1) - \mathrm{Re}(\lambda_2)$.
- $i = 9$: Local bifurcation (zero eigenvalue or Hopf): number of stable eigenvalues decreases; $\mathrm{Re}(\mu_1) = 0$.
- $i = 10$: Local bifurcation (zero eigenvalue or Hopf): number of unstable eigenvalues decreases; $\mathrm{Re}(\lambda_1) = 0$.
- $i = 11$: 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.c.

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.*

15.7 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 `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 `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 $U = ...(T)$ 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

$$\xi_1^2 + \xi_2^2 + \ldots + \xi_{\texttt{NUNSTAB}}^2 = \epsilon_0^2,$$

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)`	:	ϵ_0
`PAR(IP+i)`	:	ξ_i, `i=1,2,...,NUNSTAB`
`PAR(IP+NUNSTAB+i)`	:	ω_i, `i=1,2,...,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 18 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.

15.8 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 15.1 and Table 15.2 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 7.4. 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
ld('san')	load the problem defition
run(c='san.1',h='san.1')	get the HomCont constants-file and run AUTO /HomCont
sv('6')	save output-files as `b.6, s.6, d.6`

Table 15.1: An example of AUTO -Commands.

COMMAND	ACTION
run(c='san.9',h='san.9',s='6')	get the HomCont constants-file and run AUTO /HomCont; restart solution read from `s.6`
ap('6')	append output-files to `b.6, s.6, d.6`

Table 15.2: Another example of AUTO -Commands.

Chapter 16

HomCont Demo : san.

16.1 Sandstede's Model.

Consider the system (Sandstede 1995*a*)

$$\begin{array}{rcl} \dot{x} & = & a\,x + b\,y - a\,x^2 + (\tilde{\mu} - \alpha\,z)\,x\,(2-3x) \\ \dot{y} & = & b\,x + a\,y - \frac{3}{2}\,b\,x^2 - \frac{3}{2}\,a\,x\,y - (\tilde{\mu} - \alpha\,z)\,2\,y \\ \dot{z} & = & c\,z + \mu\,x + \gamma\,x\,y + \alpha\,\beta\,(x^2\,(1-x) - y^2) \end{array} \tag{16.1}$$

as given in the file `san.c`. Choosing the constants appearing in (16.1) appropriately allows for computing inclination and orbit flips as well as non-orientable resonant bifurcations, see (Sandstede 1995*a*) for details and proofs. The starting point for all calculations is $a = 0$, $b = 1$ where there exists an explicit solution given by

$$(x(t), y(t), z(t)) = \left(1 - \left(\frac{1-e^t}{1+e^t}\right)^2, 4\,e^t\,\frac{1-e^t}{(1+e^t)^3}, 0\right).$$

This solution is specified in the routine **stpnt**.

16.2 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 $(a, \tilde{\mu})$ `=(PAR(1), 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 $(\alpha, \tilde{\mu})$ `=(PAR(4), 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 1995*a*). 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 $\epsilon =$ `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 branch 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 seems 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. 16.1. 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.

16.3 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 $a =$ `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 `PAR(21)`, implying that a non-orientable resonant bifurcation occurred at that point.

16.4 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 $(\mu, \tilde{\mu}) = ($ `PAR(7),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. 16.2).

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`.

16.5 Detailed AUTO -Commands.

AUTO -COMMAND	ACTION

`! mkdir san`
`cd san`
`demo('san')`
	create an empty work directory
	change directory
	copy the demo files to the work directory

`run(c='san.1',h='san.1')`
`sv('1')`
	continuation in `PAR(1)`
	save output-files as b.1, s.1, d.1

`run(c='san.2',h='san.2',s='1')`
`sv('2')`
	generate adjoint variables; restart from s.1
	save output-files as b.2, s.2, d.2

`run(c='san.3',h='san.3',s='2')`
`sv('3')`
	continue homoclinic orbit and adjoint; restart from s.2
	save output-files as b.3, s.3, d.3

`run(c='san.4',h='san.4',s='1')`
`sv('4')`
	no convergence without dummy step; restart from s.1
	save output-files as b.4, s.4, d.4

`run(c='san.5',h='san.5',s='3')`
`sv('5')`
	continue non-orientable orbit; restart from s.3
	save output-files as b.5, s.5, d.5

Table 16.1: Detailed AUTO -Commands for running demo san.

AUTO -COMMAND	ACTION

`run(c='san.6',h='san.6',s='san')`
`sv('6')`
	restart and homotopy to `PAR(4)`=1.0
	save output-files as b.6, s.6, d.6

`run(c='san.7',h='san.7',s='6')`
`ap('6')`
	homotopy to `PAR(5)`=0.0; restart from s.6
	append output-files to b.6, s.6, d.6

`run(c='san.8',h='san.8',s='6')`
`ap('6')`
	homotopy to `PAR(1)`=0.5; restart from s.6
	append output-files to b.6, s.6, d.6

`run(c='san.9',h='san.9',s='6')`
`ap('6')`
	homotopy to `PAR(2)`=3.0; restart from s.6
	append output-files to b.6, s.6, d.6

`run(c='san.10',h='san.10',s='6')`
`ap('6')`
	homotopy to `PAR(7)`=0.25; restart from s.6
	append output-files to b.6, s.6, d.6

`run(c='san.11',h='san.11',s='6')`
`sv('11')`
	continue in `PAR(7)` to detect orbit flip; restart from s.6
	save output-files as b.11, s.11, d.11

`run(c='san.12',h='san.12',s='11')`
`sv('12')`
	three-parameter continuation of orbit flip; restart from s.11
	save output-files as b.12, s.12, d.12

Table 16.2: Detailed AUTO -Commands for running demo san.

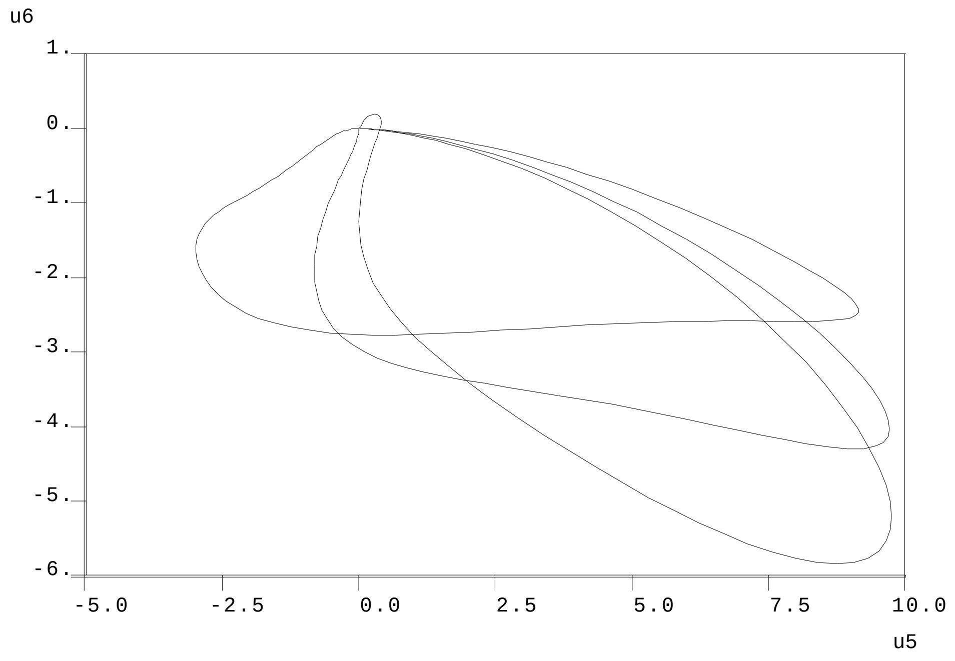


Figure 16.1: Second versus third component of the solution to the adjoint equation at labels 5, 7 and 9

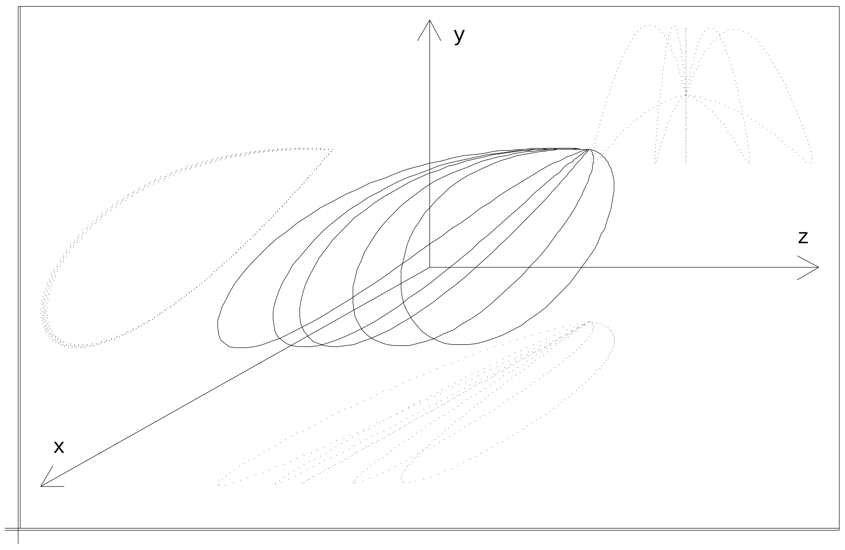


Figure 16.2: Orbits on either side of the orbit flip bifurcation. The critical orbit is contained in the (x, y)-plane

Chapter 17

HomCont Demo : mtn.

17.1 A Predator-Prey Model with Immigration.

Consider the following system of two equations (Scheffer 1995)

$$\begin{aligned} \dot{X} &= RX\left(1-\frac{X}{K}\right)-\frac{A_1XY}{B_1+X}+D_0K \\ \dot{Y} &= E_1\frac{A_1XY}{B_1+X}-D_1Y-\frac{A_2ZY^2}{B_2^2+Y^2}. \end{aligned} \tag{17.1}$$

The values of all parameters except (K, Z) are set as follows :

$$R=0.5,\ A_1=0.4,\ B_1=0.6,\ D_0=0.01,\ E_1=0.6,\ A_2=1.0,\ B_2=0.5,\ D_1=0.15.$$

The parametric portrait of the system (17.1) on the (Z, K)-plane is presented in Figure 17.1. 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).

17.2 Continuation of Central Saddle-Node Homoclinics.

Local bifurcation analysis shows that at $K = 6.0,\ Z = 0.06729762\ldots$, the system has a saddle-node equilibrium

$$(X^0, Y^0) = (5.738626\ldots, 0.5108401\ldots),$$

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.c` contains approximate parameter values

$$K = \texttt{PAR(1)} = 6.0,\ Z = \texttt{PAR(2)} = 0.06729762,$$

as well as the coordinates of the saddle-node

$$X^0 = \texttt{PAR(12)} = 5.738626,\ Y^0 = \texttt{PAR(13)} = 0.5108401,$$

and the length of the truncated time-interval

$$T_0 = \texttt{PAR(11)} = 1046.178\,.$$

Since a homoclinic orbit to a saddle-node is being followed, we have also made the choices

$$\texttt{IEQUIB} = \texttt{2} \quad \texttt{NUNSTAB} = \texttt{0} \quad \texttt{NSTAB} = \texttt{1}$$

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

$$D_1 = (K^1, Z^1) = (6.6104\ldots, 0.069325\ldots)$$

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

$$D_2 = (K^2, Z^2) = (5.1803\ldots, 0.063855\ldots).$$

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.

17.3 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 branch. 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 branch of P emanating from it, however, the solutions between D_1 and D_2 should be considered as spurious[1], therefore we do not save these data. The reliable way to compute the lower branch 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 branch of P approaching the BT point (see Figure 17.1)

[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 17.1.

It is worthwhile to compare the homoclinic curves computed above with a curve $T_0 = const$ 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 17.2. 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.

17.4 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=`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 branch 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 17.3.

17.5 Detailed AUTO -Commands.

AUTO -COMMAND	ACTION
! mkdir mtn	create an empty work directory
cd mtn	change directory
demo('mtn')	copy the demo files to the work directory
us('mtn')	use the starting data in `mtn.dat` to create `s.dat`
run(c='mtn.1',h='mtn.1',s='dat')	continue saddle-node homoclinic orbit
sv('1')	save output-files as `b.1, s.1, d.1`
run(c='mtn.2',h='mtn.2',s='1')	continue in opposite direction; restart from `s.1`
ap('1')	append output-files to `b.1, s.1, d.1`
run(c='mtn.3',h='mtn.3',s='1')	switch to saddle homoclinic orbit ; restart from `s.1`
ap('1')	append output-files to `b.1, s.1, d.1`
run(c='mtn.4',h='mtn.4',s='1')	continue in reverse direction; restart from `s.1`
sv('4')	save output-files as `b.4, s.4, d.4`
run(c='mtn.5',h='mtn.5',s='1')	other saddle homoclinic orbit branch; restart from `s.1`
ap('1')	append output-files to `b., s.1, d.1`
run(c='mtn.6',h='mtn.6',s='1')	3-parameter non-central saddle-node homoclinic.
sv('6')	save output-files as `b.6, s.6, d.6`

Table 17.1: Detailed AUTO -Commands for running demo `mtn`.

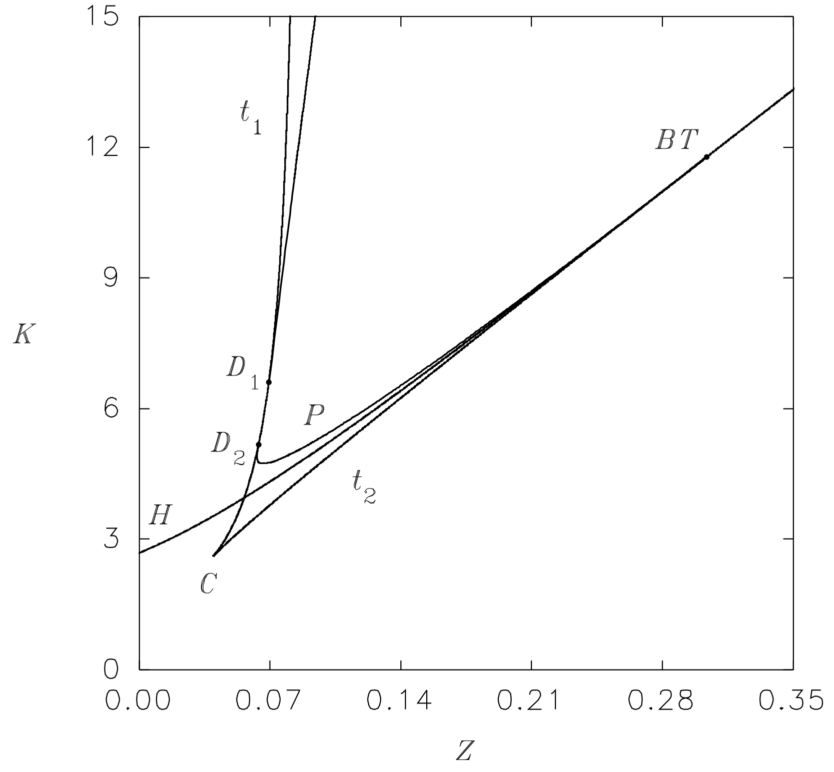


Figure 17.1: Parametric portrait of the predator-prey system

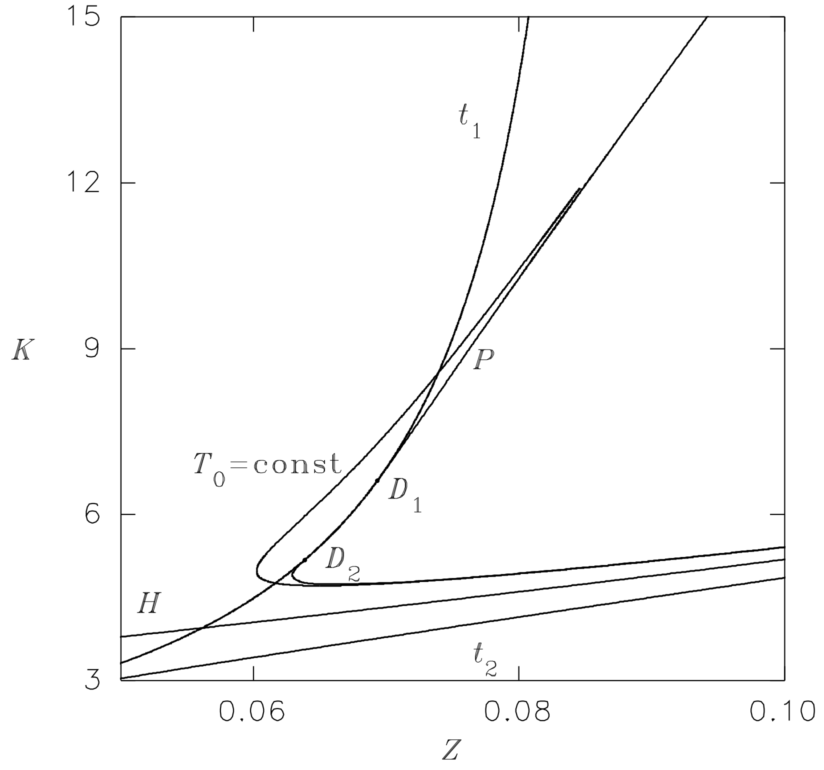


Figure 17.2: Approximation by a large-period cycle

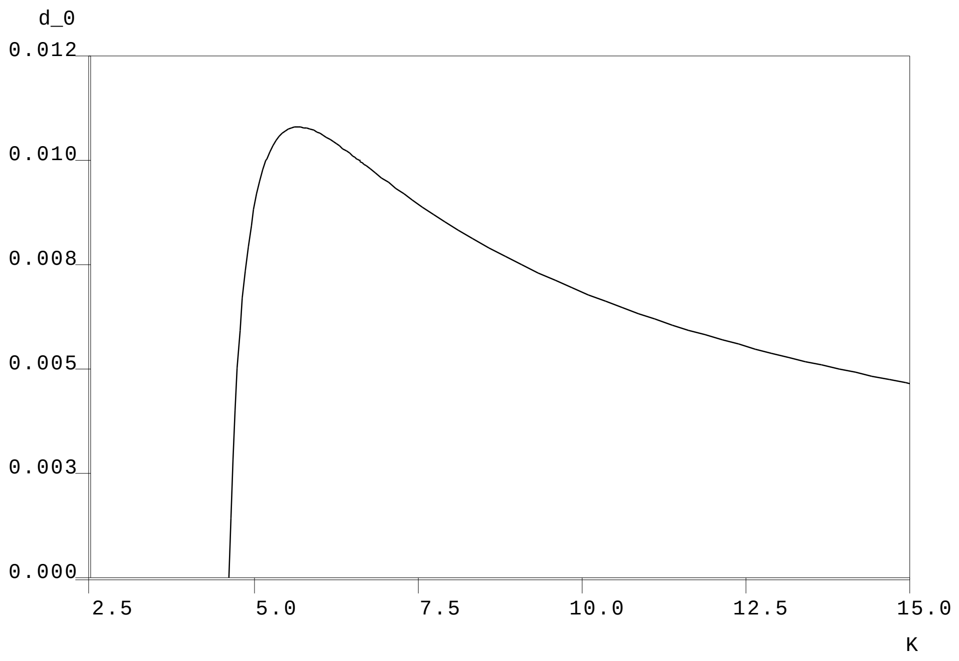


Figure 17.3: Projection onto the (K, D_0)-plane of the three-parameter curve of non-central saddle-node homoclinic orbit

Chapter 18

HomCont Demo : kpr.

18.1 Koper's Extended Van der Pol Model.

The equation-file kpr.c contains the equations

$$\begin{aligned} \dot{x} &= \epsilon_1^{-1}\,(k\,y - x^3 + 3\,x - \lambda) \\ \dot{y} &= x - 2\,y + z \\ \dot{z} &= \epsilon_2(y - z), \end{aligned} \tag{18.1}$$

with $\epsilon_1 = 0.1$ and $\epsilon_2 = 1$ (Koper 1995).

To copy across the demo kpr and compile we type

```
@dm kpr
```

18.2 The Primary Branch of Homoclinics.

First, we locate a homoclinic orbit using the homotopy method. The file kpr.c already contains approximate parameter values for a homoclinic orbit, namely $\lambda =$ `PAR(1)=-1.851185`, $k =$ `PAR(2)=-0.15`. The files c.kpr.1 and h.kpr.1 specify the appropriate constants for continuation in $2T$ `=PAR(11)` (also referred to as `PERIOD`) and the dummy parameter ω_1= `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 18.1) it can be noted that although the right-hand projection boundary condition is satisfied, the solution is still quite away from the equilibrium.

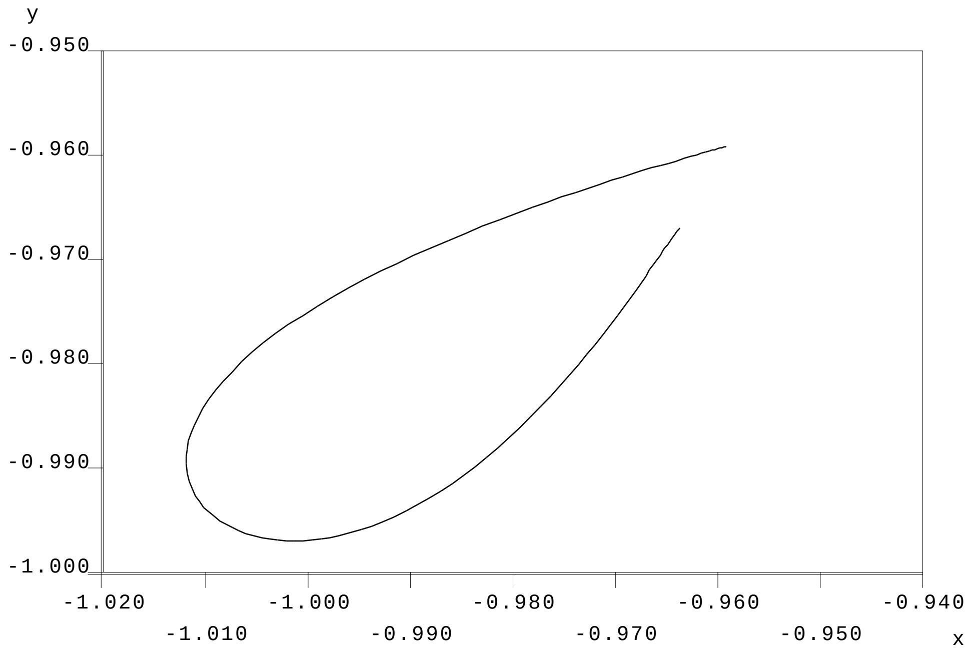


Figure 18.1: Projection on the (x, y)-plane of solutions of the boundary value problem with $2T = 19.08778$.

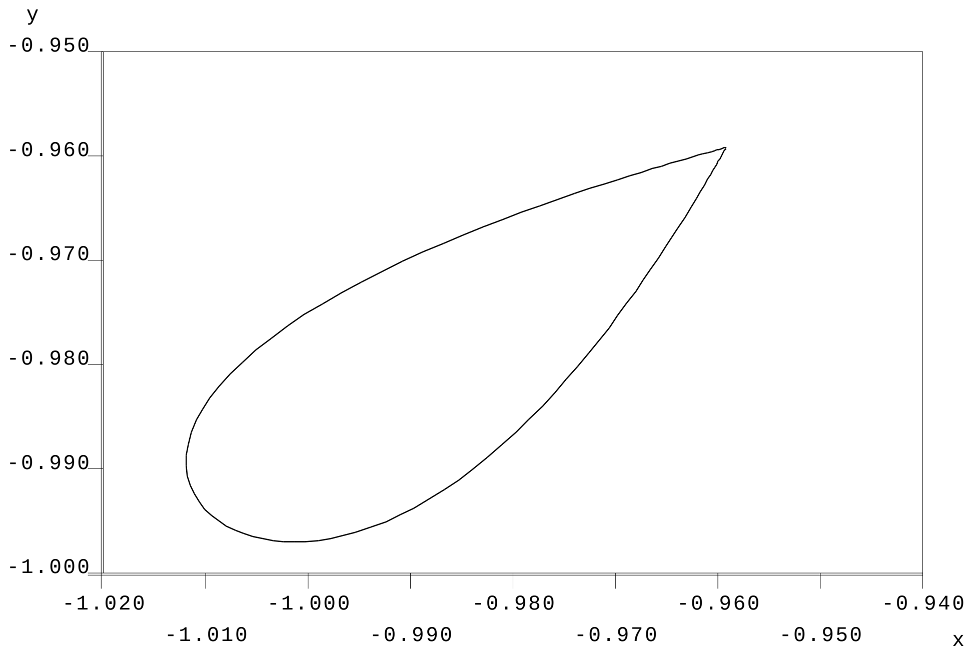


Figure 18.2: Projection on the (x, y)-plane of solutions of the boundary value problem with $2T = 60.0$.

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 18.2).

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 15.6.

Note that at the end-point of the branch (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 18.3 presents solutions $\phi(t)$ of the modified adjoint variational equation (for details see Champneys et al. (1996)) at parameter values on the homoclinic branch before and after the first detected inclination

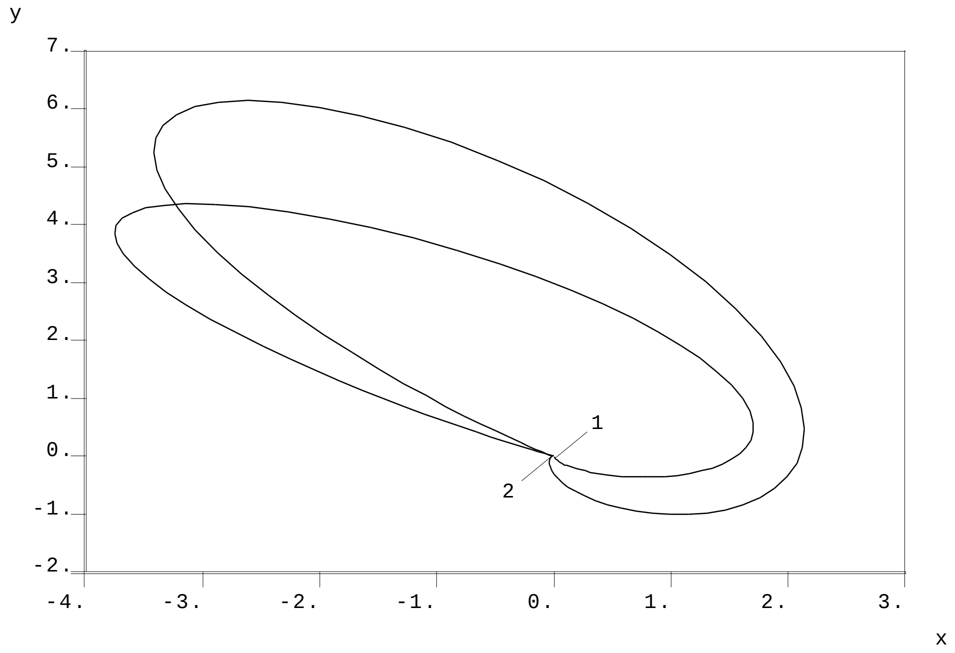


Figure 18.3: Projection on the (x, y)-plane of solutions $\phi(t)$ at `1` ($\lambda = -1.825470, k = -0.1760749$) and `2` ($\lambda = -1.686154, k = -0.3183548$).

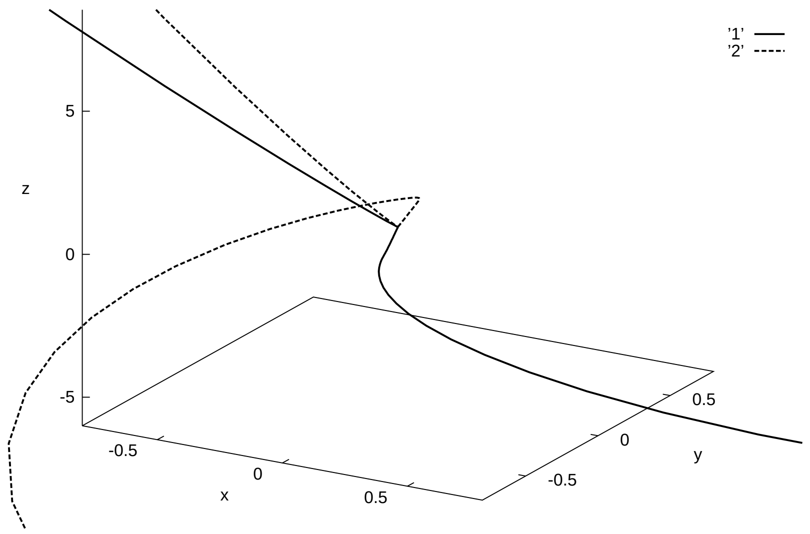


Figure 18.4: Three-dimensional blow-up of the solution curves $\phi(t)$ at labels `1` (dotted) and `2` (solid line) from Figure 3.8.

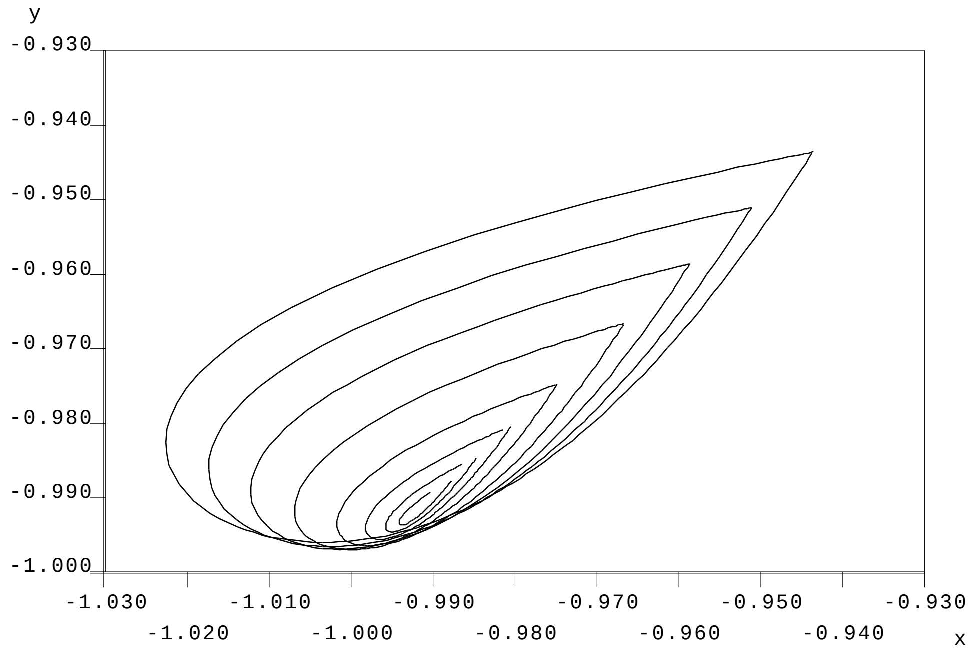


Figure 18.5: Computed homoclinic orbits approaching the BT point

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 18.4. 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 18.5. 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.

18.3 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 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 branch 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 branch 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 18.6 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 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.

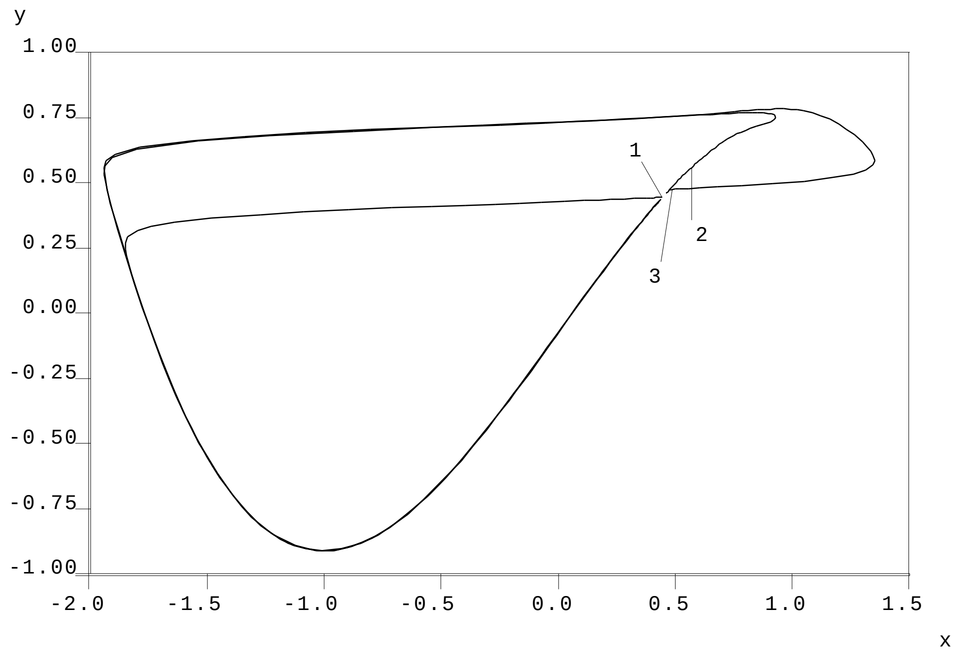


Figure 18.6: Two non-central saddle-node homoclinic orbits, 1 and 3; and, 2, a central saddle-node homoclinic orbit between these two points

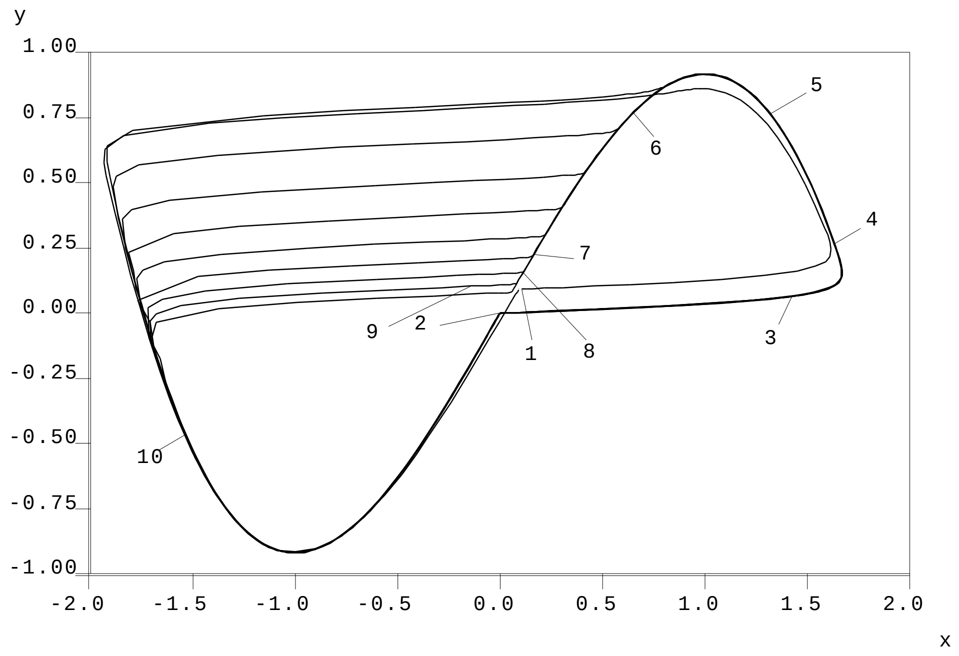


Figure 18.7: The big homoclinic orbit approaching a figure-of-eight

`make ninth`

Note that AUTO takes a large number of steps near the line `PAR(1)=0`, while `PAR(2)` approaches $-2.189\ldots$ (which is why we chose such a large value `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 18.7) 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 `PAR(1)=0` corresponds to a symmetry in the differential equations (Koper 1994); note also that the equilibrium, stored as (`PAR(12)`, `PAR(13)`, `PAR(14)`) in `d.9`, approaches the origin as we approach the figure-of-eight homoclinic.

18.4 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 $\epsilon =$ `PAR(3)`. First we continue the first inclination flip stored at label `7` in `s.3`

make tenth

Note that `ITWIST=1` in `h.kpr.10`, so that the adjoint is also continued, and there is one fixed condition `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.

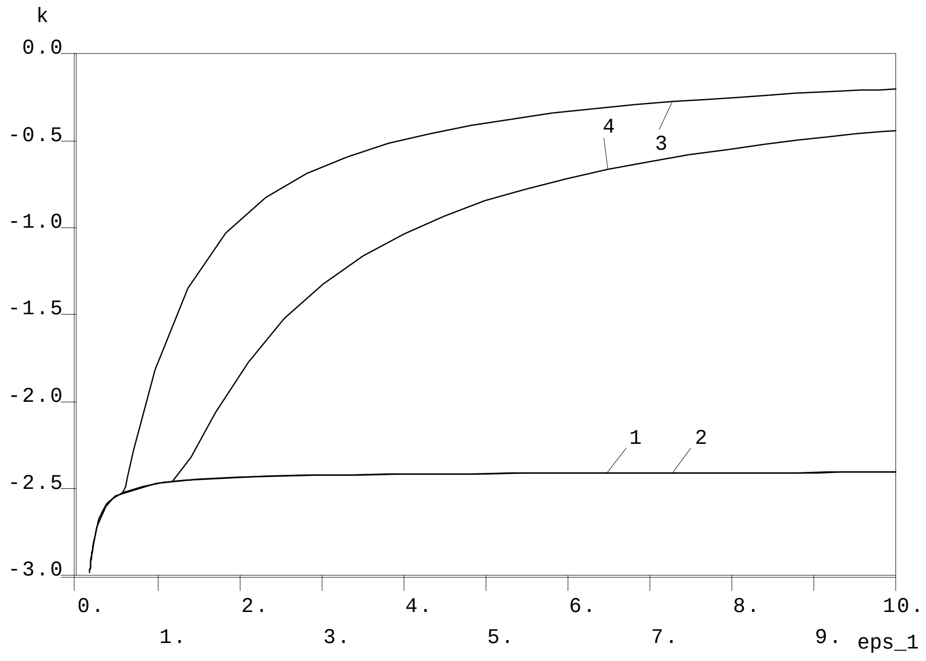
```
make twelfth
make thirteenth
```

The projection onto the (ϵ, k)-plane of all four of these codimension-two curves is given in Figure 18.8. 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 18.6 the orbits look quite distinct in phase space.

18.5 Detailed AUTO -Commands.

AUTO -COMMAND	ACTION
`! mkdir kpr`	create an empty work directory
`cd kpr`	change directory
`demo('kpr')`	copy the demo files to the work directory
`run(c='kpr.1',h='kpr.1')`	continuation in the time-length parameter `PAR(11)`
`sv('1')`	save output-files as b.1, s.1, d.1
`run(c='kpr.2',h='kpr.2',s='1')`	locate the homoclinic orbit; restart from s.1
`sv('2')`	save output-files as b.2, s.2, d.2
`run(c='kpr.3',h='kpr.3',s='2')`	generate adjoint variables ; restart from s.2
`sv('3')`	save output-files as b.3, s.3, d.3
`run(c='kpr.4',h='kpr.4',s='3')`	continue the homoclinic orbit; restart from s.3
`ap('3')`	append output-files to b.3, s.3, d.3
`run(c='kpr.5',h='kpr.5',s='3')`	continue in reverse direction; restart from s.3
`ap('3')`	append output-files to b.3, s.3, d.3
`run(c='kpr.6',h='kpr.6',s='2')`	increase the period; restart from s.2
`sv('6')`	save output-files as b.6, s.6, d.6

Table 18.1: Detailed AUTO -Commands for running demo kpr.



.

Figure 18.8: Projection onto the (PAR(3),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

`run(c='kpr.7',h='kpr.7',s='6')`
`sv('7')`
	recompute the branch of homoclinic orbits; restart from s.6
	save output-files as b.7, s.7, d.7

`run(c='kpr.8',h='kpr.8',s='7')`
`sv('8')`
	continue central saddle-node homoclinics; restart from s.7
	save output-files as b.8, s.8, d.8

`run(c='kpr.9',h='kpr.9',s='8')`
`sv('9')`
	continue homoclinics from codim-2 point; restart from s.8
	save output-files as b.9, s.9, d.9

`run(c='kpr.10',h='kpr.10',s='3')`
`sv('10')`
	3-parameter curve of inclination-flips; restart from s.3
	save output-files as b.10, s.10, d.10

`run(c='kpr.11',h='kpr.11',s='3')`
`sv('11')`
	another curve of inclination-flips; restart from s.3
	save output-files as b.11, s.11, d.11

`run(c='kpr.12',h='kpr.12',s='7')`
`sv('12')`
	continue non-central saddle-node homoclinics; restart from s.7
	save output-files as b.12, s.12, d.12

`run(c='kpr.13',h='kpr.13',s='8')`
`ap('12')`
	continue non-central saddle-node homoclinics; restart from s.8
	append output-files to b.12, s.12, d.12

Table 18.2: Detailed AUTO -Commands for running demo kpr.

Chapter 19

HomCont Demo : cir.

19.1 Electronic Circuit of Freire *et al.*

Consider the following model of a three-variable electronic circuit (Freire, Rodríguez-Luis, Gamero & Ponce 1993)

$$\left\{ \begin{array}{rcl} \dot{x} &=& \left[-(\beta+\nu)x + \beta y - a_3 x^3 + b_3 (y-x)^3\right]/r, \\ \dot{y} &=& \beta x - (\beta+\gamma) y - z - b_3 (y-x)^3, \\ \dot{z} &=& y. \end{array} \right. \tag{19.1}$$

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.c` 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 $2T =$ `PAR(11)`$= 36.13$. We wish to follow the branch 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 $\nu =$ `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 branch of saddle-focus homoclinic orbits, viz. ψ_2, ψ_4, ψ_5, ψ_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 `PAR(25)`≈ 0, corresponds to a neutrally-divergent saddle-focus, $\psi_5 = 0$. Label 7, where `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 `PAR(2)` is also approximately zero at label 7, which accords with the analytical observation that the origin of (19.1) 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. 19.1. 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 branch. 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 branch are plotted in Fig. 19.2. If we had continued further (by increasing `NMX`), the computation would end at a no convergence error `TY=MX` owing to the homoclinic branch 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)`.

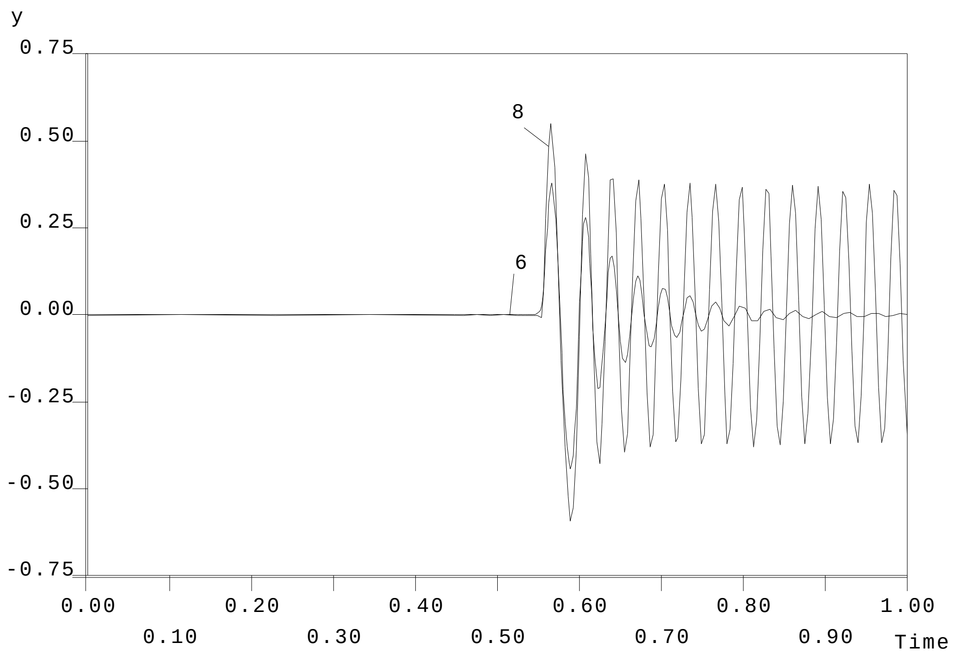


Figure 19.1: Solutions of the boundary value problem at labels 6 and 8, either side of the Shil'nikov-Hopf bifurcation

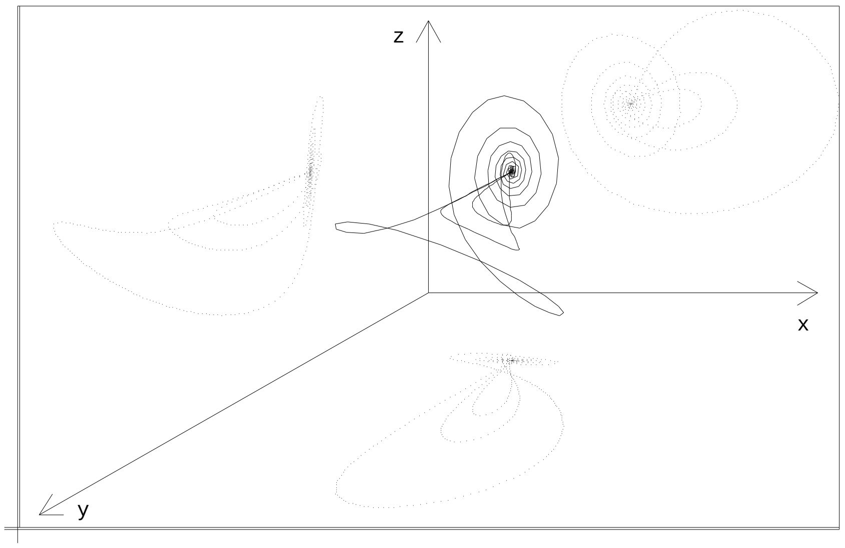


Figure 19.2: Phase portraits of three homoclinic orbits on the branch, showing the saddle-focus to saddle transition

19.2 Detailed AUTO -Commands.

AUTO -COMMAND	ACTION
`! mkdir cir`	create an empty work directory
`cd cir`	change directory
`demo('cir')`	copy the demo files to the work directory
`us('cir')`	use the starting data in cir.dat to create s.dat
`run(c='cir.1',h='cir.1',s='dat')`	increase the truncation interval; restart from s.dat
`sv('1')`	save output-files as b.1, s.1, d.1
`run(c='cir.2',h='cir.2',s='1')`	continue saddle-focus homoclinic orbit; restart from s.1
`sv('2')`	save output-files as b.2, s.2, d.2
`run(c='cir.3',h='cir.3',s='2')`	generate adjoint variables ; restart from s.2
`ap('2')`	append output-files as b.2, s.2, d.2

Table 19.1: Detailed AUTO -Commands for running demo cir.

Chapter 20

HomCont Demo : she.

20.1 A Heteroclinic Example.

The following system of five equations Rucklidge & Mathews (1995)

$$\begin{aligned}
\dot{x} &= \mu\, x + x\, y - z\, u, \\
\dot{y} &= -y - x^2, \\
\dot{z} &= (4\sigma\, x\, u + 4\sigma\, \mu\, z - 9\sigma\, z + 4x\, u + 4\mu\, z)/4(1+\sigma) \\
\dot{u} &= -\sigma u/4 - \sigma Q v/4\pi^2 + 3(1+\sigma)xz/4\sigma \\
\dot{v} &= \zeta u/4 - \zeta v/4
\end{aligned} \tag{20.1}$$

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.c`[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
```

[1]The last parameter used to store the equilibria (`PAR(21)`) is overlaped 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.c`. 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 branch. Finally, we can continue in the opposite direction along the branch 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 20.1, from which we see that the orbit shrinks to zero as `PAR(1)`$=\mu \to 0$.

20.2 Detailed AUTO -Commands.

AUTO -COMMAND	ACTION
`! mkdir she`	create an empty work directory
`cd she`	change directory
`demo('she')`	copy the demo files to the work directory
`us('she')`	use the starting data in she.dat to create s.dat
`run(c='she.1',h='she.1',s='dat')`	continue heteroclinic orbit; restart from s.dat
`sv('1')`	save output-files as b.1, s.1, d.1
`run(c='she.2',h='she.2',s='dat')`	repeat with `IEQUIB=-1`
`sv('2')`	save output-files as b.2, s.2, d.2
`run(c='she.3',h='she.3',s='2')`	continue in reverse direction ; restart from s.2
`ap('2')`	append output-files to b.2, s.2, d.2

Table 20.1: Detailed AUTO -Commands for running demo she.

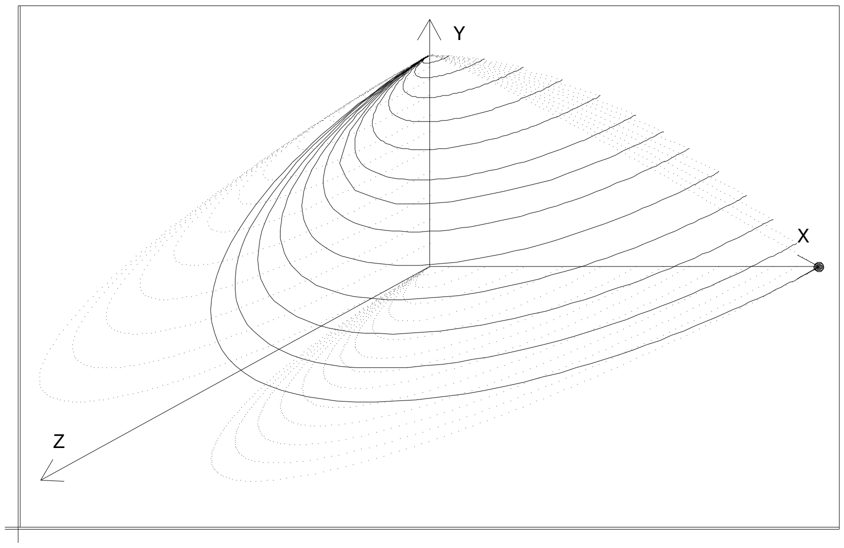


Figure 20.1: Projections into (x, y, z)-space of the family of heteroclinic orbits.

Chapter 21

HomCont Demo : rev.

21.1 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

$$\begin{cases} \dot{u_1} &= u_2 \\ \dot{u_2} &= u_3 \\ \dot{u_3} &= u_4 \\ \dot{u_4} &= -Pu_3 - u_1 + u_1^3 \end{cases} \tag{21.1}$$

Note that (21.1) is invariant under two separate reversibilities

$$R_1 : (u_1, u_2, u_3, u_4, t) \mapsto (u_1, -u_2, u_3, -u_4, -t) \tag{21.2}$$

and

$$R_2 : (u_1, u_2, u_3, u_4, t) \mapsto (-u_1, u_2, -u_3, u_4, -t) \tag{21.3}$$

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.

21.2 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.c and rev.dat. The orbit contained in the data file is a "primary" homoclinic solution for $P = 1.6$, with truncation (half-)interval `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 $P \to 0$. 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 21.1.

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 21.2.

21.3 An R_2-Reversible Homoclinic Solution.

```
make third
```

Copies the files rev.c.3 and rev.dat.3 to rev.c 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 `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
```

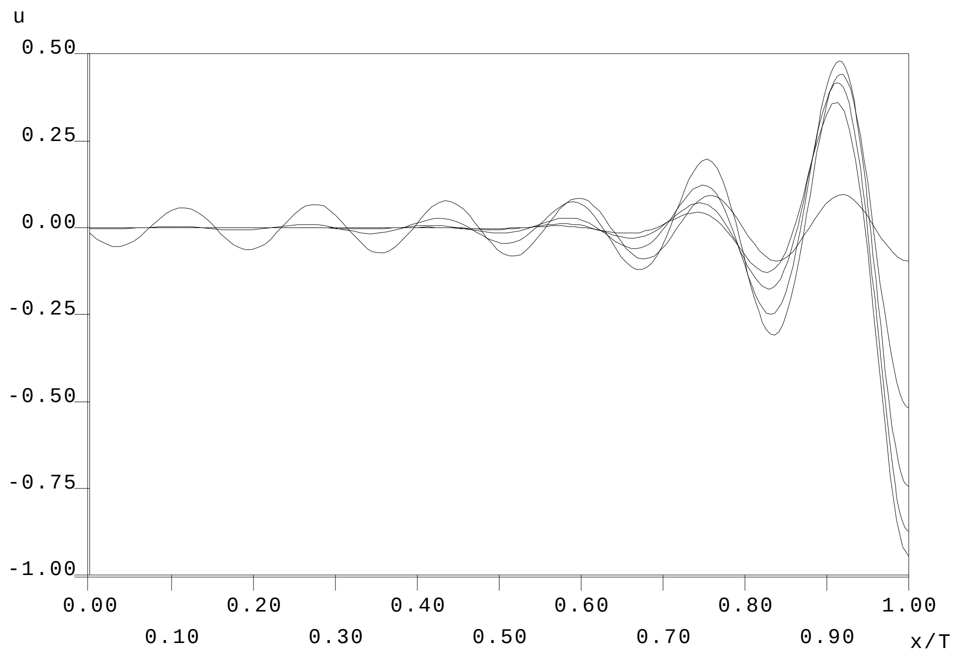


Figure 21.1: R_1-Reversible homoclinic solutions on the half-interval $x/T \in [0, 1]$ where $T = 39.0448429$ for P approaching 2 (solutions with labels `1-5` respectively have decreasing amplitude)

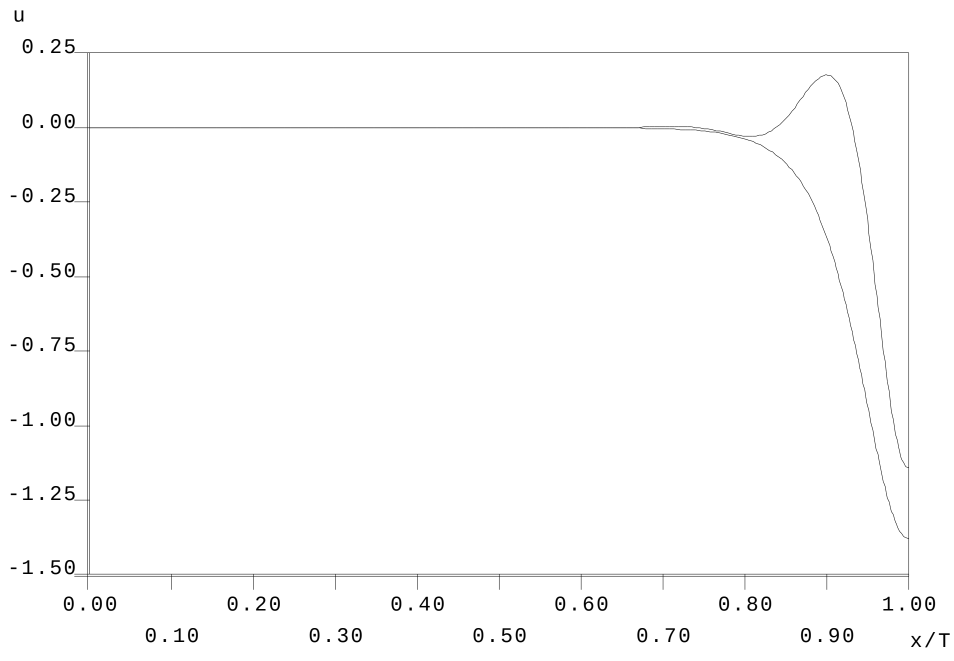


Figure 21.2: R_1-reversible homoclinic orbits with oscillatory decay as $x \to -\infty$ (corresponding to label `6`) and monotone decay (at label `10`)

```
   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 21.3. The computation ends in a no-convergence point. The solution here is depicted in Figure 21.4. The lack of convergence is due to the large peak and trough of the solution rapidly moving to the left as $P \to -2$ (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.

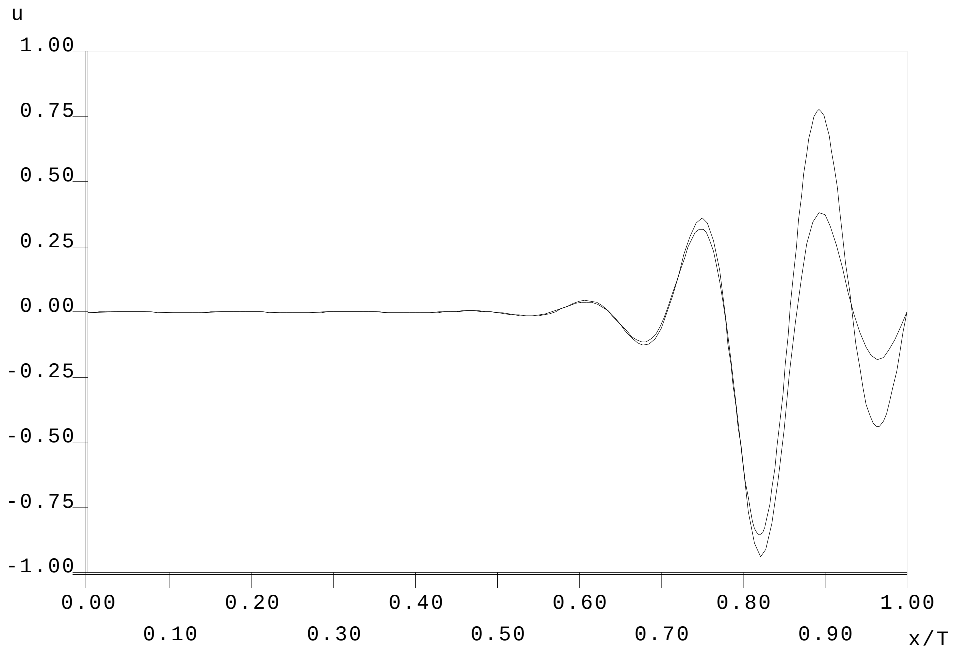


Figure 21.3: Two R_2-reversible homoclinic orbits at $P = 1.6$ corresponding to labels `1` (smaller amplitude) and `5` (larger amplitude)

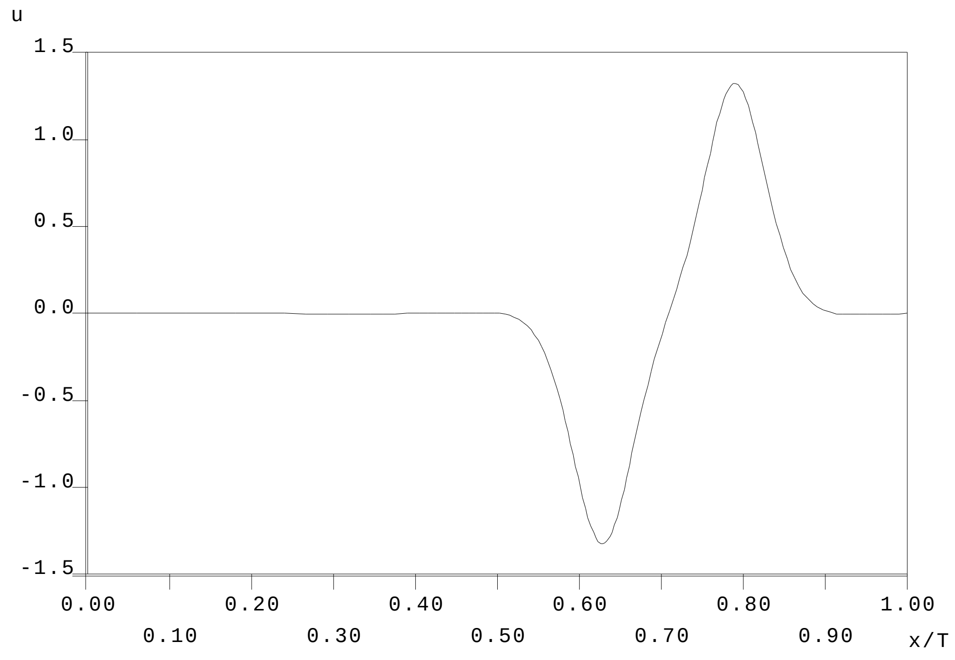


Figure 21.4: An R_2-reversible homoclinic orbit at label `8`

21.4 Detailed AUTO -Commands.

AUTO -COMMAND	ACTION
`! mkdir rev`	create an empty work directory
`cd rev`	change directory
`demo('rev')`	copy the demo files to the work directory
`cp rev.c.1 rev.c`	get equations file to rev.c
`cp rev.dat.1 rev.dat`	get the starting data to rev.dat
`us('rev')`	use the starting data in rev.dat to create s.dat
`run(c='rev.1',h='rev.1',s='dat')`	increase `PAR(1)`
`sv('1')`	save output-files as b.1, s.1, d.1
`run(c='rev.2',h='rev.2',s='1')`	continue in reverse direction; restart from s.1
`ap('1')`	append output-files to b.1, s.1, d.1
`cp rev.c.3 rev.c`	get equations file with new value of `PAR(11)`
`cp rev.dat.3 rev.dat`	get starting data with different reversibility
`us('rev')`	use the starting data in rev.dat to create s.dat
`run(c='rev.3',h='rev.3',s='dat')`	restart with different reversibility
`sv('3')`	save output-files as b.3, s.3, d.3
`run(c='rev.4',h='rev.4',s='3')`	continue in reverse direction; restart from s.3
`ap('3')`	append output-files to b.3, s.3, d.3

Table 21.1: Detailed AUTO -Commands for running demo rev.

Chapter 22

HomCont Demo : Homoclinic branch switching.

This demo illustrates homoclinic branch switching, which is an implementation of Lin's method (Lin 1990, Sandstede 1993, C. 2001) as described in Oldeman et al. (2001). 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 16. 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.

22.1 Branch switching at an inclination flip in Sandstede's model.

Consider the system (Sandstede 1995*a*)

$$\begin{array}{rcl} \dot{x} & = & ax + by - ax^2 - \alpha zx(2-3x), \\ \dot{y} & = & bx + ay - \frac{3}{2}x(bx+ay) + \alpha z2y, \\ \dot{z} & = & cz + \mu x + 3xz + \alpha(x^2(1-x) - y^2). \end{array} \tag{22.1}$$

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  0.000000E+00 ... 6.666667E-01  0.000000E+00  0.000000E+00
  1     6  EP   2  3.750000E-01 ... 6.666667E-01 -1.333333E-01  0.000000E+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    18  HB   3  3.184290E-01 ... 6.543750E-01 -1.347543E-01  7.701025E-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 μ=`PAR(5)` and `PAR(10)`, 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     5       5 -2.418809E-03  6.705689E-01  ...  1.089749E+01
                                                ...
  3    40       8 -1.294950E-02  6.145469E-01  ...  1.412970E+01
                                                ...
  3    81  EP  13 -1.046566E-04  4.018291E-01  ...  3.499999E+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 22.1(a), where the periodic orbit grows from the Hopf equilibrium to a homoclinic orbit.

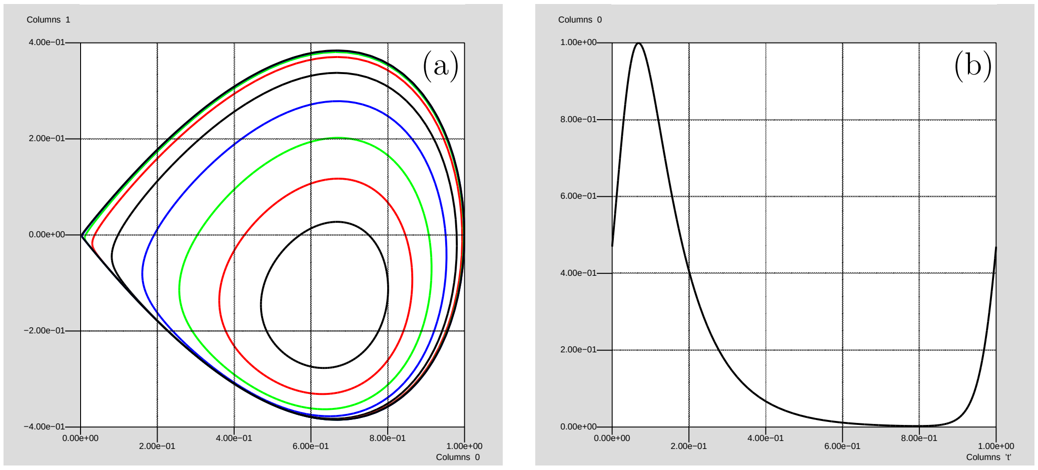


Figure 22.1: Periodic orbit growing from a Hopf bifurcation to a homoclinic orbit (a). The unshifted homoclinic orbit (b).

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 22.1(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     15   EP   14 -2.000000E+00   4.018899E-01     ...   2.661459E-09
```

The output is saved in the files b.4, s.4 and d.4. Note that `PAR(5)`$=\mu$ 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 is 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)
chc("ITWIST",1)
chc("ISTART",1)
rn(s='4')
sv('5')
```

where `chc` 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   2.550843E-09  4.018898E-01 ...  -1.000000E-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(32)`.

```
rn(c='sib.6',h='sib.if',s='5')
sv('6')
```

```
 BR    PT  TY LAB    PAR(4)         L2-NORM      ...      PAR(5)         PAR(32)
  3    11  UZ  16  7.117745E-02  4.018899E-01   ...   1.243774E-11 -2.366987E-07
```

The output is stored in b.6, s.6 and d.6. Hence an inclination flip was found at $\alpha = 0.7117745$.

Now we are ready to perform homoclinic branch switching, using the techniques described in (Oldeman et al. 2001). 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 15 that the Lin gaps ε_i correspond to `PAR(19+i*2)` and the time intervals T_i correspond to `PAR(20+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(20)      L2-NORM     ...     PAR(21)       PAR(5)
 3    10      18  3.458968E+01  4.468176E-01  ...   7.877123E-07 -1.558861E-11
 3    20      19  2.736992E+01  4.468176E-01  ...   2.911187E-05 -1.639739E-09
 3    30      20  1.737196E+01  4.468171E-01  ...   4.422734E-03 -3.101671E-05
 3    38  EP  21  1.014512E+01  4.467963E-01  ...   2.000000E-01 -1.486151E-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 22.2. Next we are ready to close this gap, by continuing in α, μ, and ε_1, while keeping T_1 at a constant value.

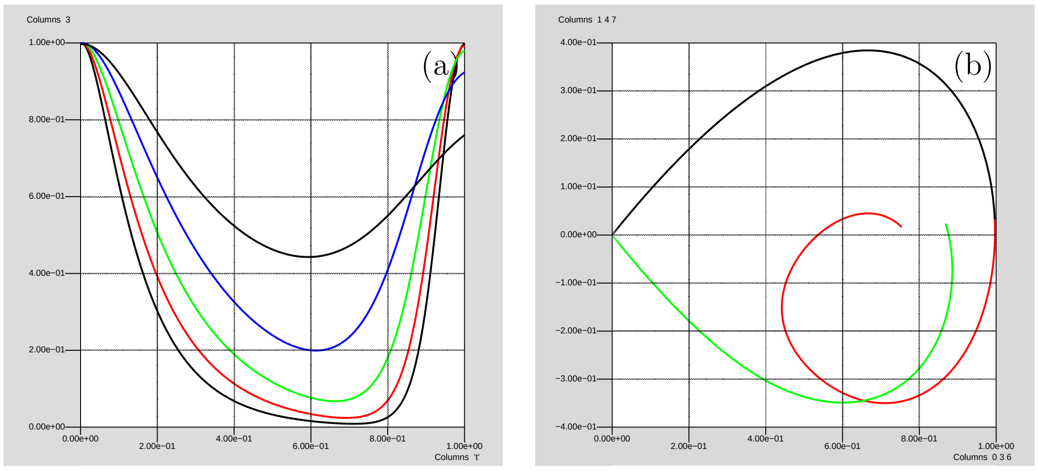


Figure 22.2: Behaviour of the second piece of the 'broken homoclinic orbit' when creating a Lin gap (a). Projection of the "broken homoclinic orbit" onto the (x, y)-plane, where $\varepsilon_1 = 0.2$. To include all the pieces necessary to obtain this figure, the "X" box must contain [0,3,6] and the "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(21)
 3     3  UZ  22  7.399999E-02  4.467807E-01  ...  -1.431624E-02  1.937464E-01
 3    32  EP  23  1.992281E-01  4.465901E-01  ...  -6.054949E-03  2.292996E-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.8',h='sib.hom',s='6')
ap('6')
```

```
BR    PT  TY LAB    PAR(4)       L2-NORM     ...     PAR(5)
 3     7  UZ  24  1.499999E-01  4.944903E-01  ...  -3.602482E-03
 3    30  EP  25  7.614033E-02  4.987463E-01  ...  -2.648395E-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 22.3(a).

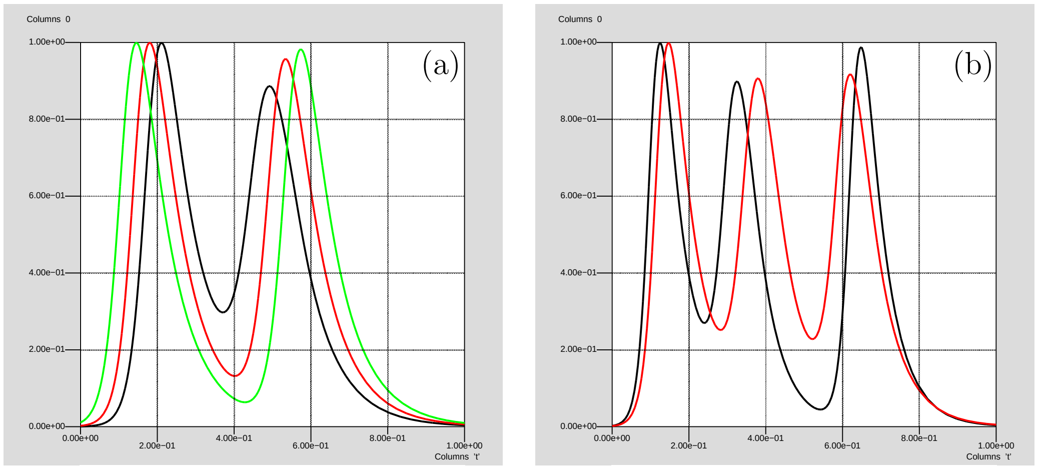


Figure 22.3: The 2-homoclinic orbit as a is changed (a). The two different 3-homoclinic orbits (b).

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, ε_2=`PAR(23)` and, once again, stop when ε_1=`PAR(21)=0.2`. 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(20)      ...    PAR(21)        PAR(23)        PAR(5)
  3    10       26   3.458963E+01   ... 7.878940E-07   6.421573E-07 -1.062630E-11
  3    20       27   2.736987E+01   ... 2.911260E-05   6.515911E-07 -1.636554E-09
  3    30       28   1.737189E+01   ... 4.422894E-03   1.440898E-04 -3.101882E-05
  3    38  EP   29   1.014512E+01   ... 2.000000E-01   6.974453E-02 -1.486151E-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(21)        PAR(23)
  3     6  UZ  30  8.199998E-02  ... -1.297904E-02  1.769949E-01  6.371836E-02
  3    32  EP  31  1.984145E-01  ... -6.054949E-03  2.307164E-06  3.624489E-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(22)`) 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(22)        PAR(23)
  3    16  UZ  32  1.983953E-01  ... -6.055361E-03  2.013107E+01  1.824909E-08
  3    24  UZ  33  1.800000E-01  ... -6.502928E-03  1.275539E+01 -3.142935E-02
  3    30  UZ  34  1.669900E-01  ... -6.892692E-03  9.417449E+00 -1.031790E-06
  3    32  EP  35  1.781716E-01  ... -6.553641E-03  9.502999E+00 -7.203666E-02
```

The output is appended to b.6, s.6 and d.6. Note that we have found two zeros of `PAR(23)`, 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 22.3(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  1.299993E-01  5.048071E-01  ... -2.339037E-03
  3    30  EP  37  9.272363E-02  5.065599E-01  ... -2.767140E-04
```

```
rn(c='sib.14',h='sib.hom',s='6')
ap('6')
```

```
BR    PT  TY LAB    PAR(4)        L2-NORM      ...    PAR(5)
  3     4  UZ  37  1.449997E-01  5.473471E-01  ... -4.794005E-03
  3    30  EP  39  8.394009E-02  5.526047E-01  ... -7.367526E-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 22.4. It is possible and straightforward to obtain $4, 5, 6, \ldots$-homoclinic orbits by extending the above strategy.

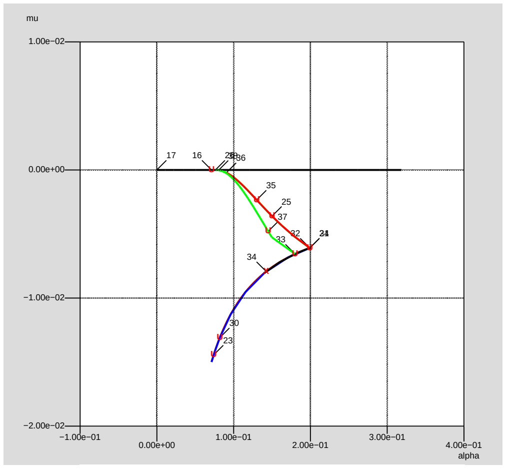


Figure 22.4: Parameter space diagram near an inclination flip. The curve through label 17 corresponds to a 1-homoclinic orbit. The opening of the Lin gaps occurs along the vertical line from label 16 to label 23. The curves through labels 23 and 30 denote the path that is followed when closing the Lin gaps. The (approximately overlaid) curves though 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 `PAR(22)`$=T_2 = 12.03201$.

22.2 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

$$\begin{aligned} u_t &= u_{xx} - f_a(u) - w, \\ w_t &= \epsilon(u - \gamma w), \end{aligned} \tag{22.2}$$

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:

$$\begin{aligned} \dot{u} &= v, \\ \dot{v} &= cv + f_a(u) + w, \\ \dot{w} &= \frac{\epsilon}{c}(u - \gamma w). \end{aligned} \tag{22.3}$$

In particular we consider solitary wave solutions of (22.2). These correspond to orbits homoclinic to $(u,v,w) = 0$ in system (22.3). 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(16)
 1    20  UZ   3  2.922565E+01  2.379162E-01  ...  -1.680003E-09
```

and a zero of `PAR(16)` 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(16)` at 0:

```
rn(c='fnb.2',h='fnb.1',s='1')
sv('2')
```

```
BR    PT  TY LAB     PERIOD       L2-NORM      ...     PAR(1)
 1   190  UZ  10  3.000000E+02  7.379317E-02  ...  1.792864E-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(1)
 1    80  UZ  13  1.000000E+03  4.041827E-02  ...  1.792865E-01
```

A continuation in `PAR(1)`$=a$ and `PAR(0)`$=c$ 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(21)` to locate Belyakov points.

```
rn(c='fnb.4',h='fnb.4',s='3')
sv('4')
```

```
BR    PT  TY LAB    PAR(1)        L2-NORM      ...     PAR(0)         PAR(21)
 1     6  UZ  15  1.318124E-01  3.287104E-02  ...  2.171656E-01 -6.312189E-06
 1    23  UZ  19 -8.545741E-08  1.561579E-02  ...  2.742181E-01 -9.887718E-02
```

Hence, there exists a Belyakov point at $(a, c) = (0.1318124, 0.217656)$. 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='fnb.5',h='fnb.5',s='4')
                            sv('5')
```

```
  BR    PT  TY LAB    PAR(8)        L2-NORM     ...    PAR(2)
   1     2  EP  28 -1.000000E+00  1.561579E-02  ...  2.500000E-03
```

Next, we continue in the time T_1 (`PAR(20)`), the gap ε_1 (`PAR(21)`) and c (`PAR(0)`), and by setting `ISTART`=-2 we try to locate a 2-homoclinic orbit:

```
                rn(c='fnb.6',h='fnb.6',s='5')
                            sv('6')
```

In fact we find many of them, exactly as is predicted by the theory:

```
  BR    PT  TY LAB    PAR(20)      ...    PAR(0)        PAR(21)
...
   1   175  UZ  45  1.647952E+02  ...  2.742181E-01 -2.313522E-11
   1   179  UZ  46  1.448063E+02  ...  2.742181E-01  1.481383E-11
   1   183  UZ  47  1.248379E+02  ...  2.742181E-01  2.171338E-16
   1   188  UZ  48  1.048192E+02  ...  2.742181E-01  5.215295E-11
   1   192  UZ  49  8.487422E+01  ...  2.742181E-01  3.106887E-15
   1   197  UZ  50  6.463349E+01  ...  2.742181E-01 -1.803730E-10
```

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 22.5. We are now in a position to continue each of these orbits

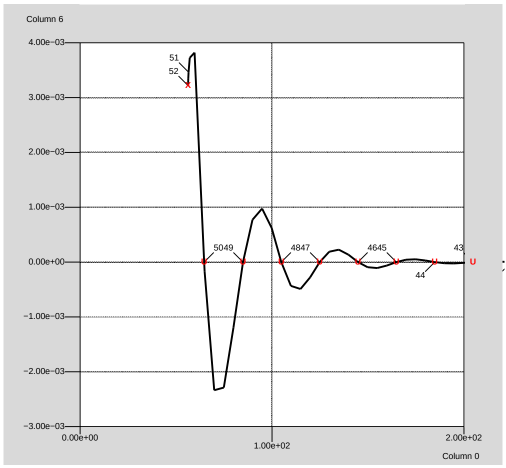


Figure 22.5: A plot of ε_1 as a function of T_1 during our computation of Shil'nikov-type two-homoclinic orbits. Each zero corresponds to a different orbit.

as a normal homoclinic orbit by setting `ISTART=1` and `ITWIST=0`. We leave this as an exercise to the reader.

22.3 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:

$$\frac{2}{15}r'''' - br'' + ar + \frac{3}{2}r^2 - \frac{1}{2}(r')^2 + [rr']' = 0. \qquad (22.4)$$

It represents solitary-wave solutions $r(x+at)$, $r \to 0$ as $x \to \pm\infty$ of the 5th-order PDE

$$r_t + \frac{2}{15}r_{xxxxx} - br_{xxx} + 3rr_x + 2r_xr_{xx} + rr_{xxx=0},$$

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 (22.4) 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

$$r(t) = 3(b + \frac{1}{2})\text{sech}^2\left([\frac{3}{4}(2b+1)]^{1/2}t\right).$$

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(0)         L2-NORM     ...    PAR(2)
  1     1  EP   1  3.000000E+00  5.565438E+00  ...  0.000000E+00
  1     2  EP   2  3.049592E+00  5.491407E+00  ...  1.807155E-17
```

Here PAR(0)=a, PAR(1)=b, and PAR(2)=λ. 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 ITWIST=1 and calculate the adjoint:

```
rn(c='kdv.2',h='kdv.2',s='1')
sv('2')
```

```
 BR    PT  TY LAB    PAR(1)         L2-NORM     ...    PAR(8)
  1     2  EP   3  2.765575E+00  5.491418E+00  ... -6.250114E-04
```

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(0)         L2-NORM     ...    PAR(2)
  1    14  UZ   5  3.000000E+00  5.476133E+00  ...  1.483821E-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 22.1; to find a 3-homoclinic orbit, we open 2 Lin gaps, until $T_1 = 3.5$, while also varying λ=PAR(2).

```
rn(c='kdv.4',h='kdv.4',s='3')
sv('4')
```

```
 BR    PT  TY LAB    PAR(2)       ...    PAR(20)       PAR(21)       PAR(23)
  1    10         8  5.797610E-10  ...  1.672717E+01 -8.381610E-08 -6.988443E-07
  1    19  UZ   9  1.399137E-09  ...  1.012493E+01  6.452744E-12  1.379764E-07
  1    20        10  2.122922E-09  ...  9.001030E+00  1.032750E-07  4.022729E-07
  1    29  EP  11  2.154196E-06  ...  3.499999E+00  7.959776E-04  3.999453E-04
```

We then look for an orbit with $a < 3$ and close the gap corresponding to ε_1=PAR(21), for decreasing a.

```
rn(c='kdv.5',h='kdv.5',s='4')
sv('5')
```

```
 BR    PT  TY LAB    PAR(1)       ...    PAR(2)        PAR(21)       PAR(23)
  1    10        12  2.579042E+00  ...  2.154861E-06  7.659464E-04  3.829183E-04
  1    13  UZ  13  2.320452E+00  ...  3.933752E-11  1.088379E-10  1.552594E-08
  1    20  EP  14 -1.906119E-01  ... -1.022044E-03 -7.600151E-01 -3.446967E-01
```

and finally close the gap corresponding to ε_2=PAR(23),

```
                    rn(c='kdv.6',h='kdv.6',s='5')
                              sv('6')

 BR    PT  TY LAB    PAR(1)        ...     PAR(2)        PAR(22)       PAR(23)
  1    23  UZ  15  2.320450E+00  ...  2.198310E-12  1.487623E+01 -4.392295E-10
  1    30      16  2.320380E+00  ... -1.004669E-09  1.027163E+01 -5.060989E-07
  1    51  UZ  17  2.336952E+00  ...  2.374866E-07  3.482932E+00  1.195914E-04
  1    58  UZ  18  3.080847E+00  ...  2.673602E-12  3.500044E+00 -1.934478E-10
  1    60  EP  19  3.134237E+00  ... -5.614124E-07  3.778288E+00 -3.398845E-04
```

so that a three-homoclinic orbit is found. Here the zero at label 17 is the one we are looking for. Label 15 is a false positive since T_2=`PAR(22)` is still too high. At label 18, a=`PAR(1)` has changed considerably to the extend that $a > 3$ and a second 3-homoclinic orbit is found. Note that for all zeros of `PAR(23)`=ε_2, the parameter λ=`PAR(2)` is also zero (within AUTO accuracy), which it has to be to remain within the original Hamiltonian system. Setting `ISTART=1`, a normal "trivial" continuation (with `NMX=1`) of the orbit corresponding to label 17 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(1)         L2-NORM     ...     PAR(2)
  1     2  EP  20  2.336952E+00  7.505830E+00 ...  2.374866E-07
```

This 3-homoclinic orbit is depicted in Figure 22.6.

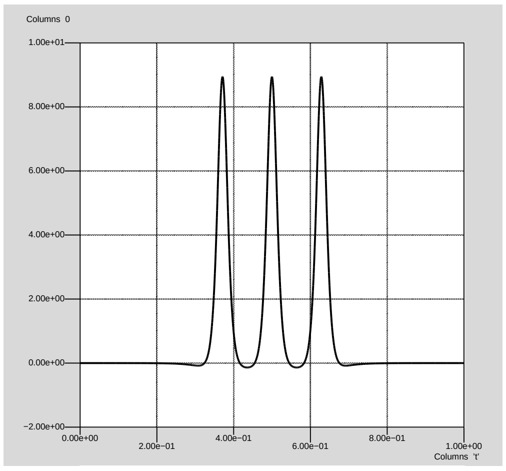


Figure 22.6: A 3-homoclinic orbit in a 5th-order Hamiltonian Korteweg-De Vries model.

Appendix A

Running AUTO using Command Mode.

AUTO can be run with the interface described in Chapter 4 or with the commands described below. The AUTO aliases must have been activated; see Section 1.2; and an equations-file `xxx.c` and a corresponding constants-file `c.xxx` (see Section 3.1) must be in the current user directory. *Do not run AUTO in the directory* `auto/2000` *or in any of its subdirectories.*

A.0.1 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.c` 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.c`, 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.c`, 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.c`, 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.

A.0.2 Plotting commands.

`@p` : Type `@p xxx` to run the graphics program PLAUT (See Chapter B) 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.

`@pr` : Type `@pr fig.x` to convert a saved PLAUT figure fig.x from compact PLOT10 format to PostScript format and send it to the printer. The converted file is called fig.x.ps. The original file is left unchanged.

A.0.3 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.

A.0.4 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 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.)

A.0.5 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.

A.0.6 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 5.3.1 and 5.3.2) are used to define the new mesh. For examples of using the `@fc` command see demos lor and pen.

`@94to97` : Type `@94to97 xxx` to convert an old AUTO 94 data-file s.xxx to new AUTO 97 format. The original file is backed up as ∼s.xxx. This conversion is only necessary for files from early versions of AUTO 94 .

A.0.7 HomCont commands.

`@h` : Use `@h` instead of `@r` when using HomCont, i.e., when IPS=9 (see Chapter 15). 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 h.xxx.

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

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

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

A.0.8 Copying a demo.

`@dm` : Type `@dm xxx` to copy all files from auto/2000/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.

A.0.9 Pendula animation.

`@pn` : Type `@pn xxx` to run the pendula animation program with data-file s.xxx. (On SGI machine only; see demo pen in Section 9.10 and the file auto/2000/pendula/README.)

A.0.10 Viewing the manual.

`@mn` : Use Ghostview to view the PostScript version of this manual.

Appendix B

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 the *@p* command defined in Section A. The PLAUT window (a Tektronix window) will appear, in which PLAUT commands can be entered. FIXME: This is not correct anymore For examples of using PLAUT see the tutorial demo `ab`, in particular, Sections 7.7 and 7.10. See also demo `pp2` in Section 9.3.

B.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 ouput 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` and `@pr`; see Section B.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.

B.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).

B.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.

B.4 Printing PLAUT Files.

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

`@pr` : Type *@pr fig.1* to convert a PLAUT file fig.1 to PostScript format and to print the resulting file fig.1.ps.

Appendix C

Graphical User Interface.

C.1 General Overview.

Please note: as of July 30, 2002 the GUI is being updated, so the documentation is this chapter is not being actively maintained. The old GUI is provided with this release of AUTO , but it is unsupported and may not be included in future releases.

The AUTO 97 graphical user interface (GUI) is a tool for creating and editing equations-files and constants-files; see Section 3.1 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 A 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/2000` *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 1.2. 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.

C.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 C.2.

C.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 5, 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.

C.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.

C.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.

C.2 The Menu Bar.

C.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.c`. The corresponding constants-file `c.xxx` is also loaded if it exists. The equation name `xxx` remains active until redefined.

C.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.

C.2.3 Write-button.

This pull-down menu contains the item `Write`, to write the loaded files `xxx.c` 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.

C.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.

C.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.

C.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.

C.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.

C.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.

C.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.

C.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.

C.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.c.

C.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.

C.3 Using the GUI.

AUTO -commands are described in Section A and illustrated in the demos. In Table C.1 we list the main AUTO -commands together with the corresponding GUI button.

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

Table C.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 A, this will run AUTO with the current equations-file xxx.c 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`.

C.4 Customizing the GUI.

C.4.1 Print-button.

The `Misc/Print`-button on the Menu Bar can be customized by editing the file GuiConsts.h in directory auto/2000/include.

C.4.2 GUI colors.

GUI colors can be customized by creating an X resource file. Two model files can be found in directory auto/2000/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.

C.4.3 On-line help.

The file auto/2000/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/2000/gui/Makefile by `CC = cc -Wf, -XNl20480 -O` On other machines, the maximum message length is the system defined maximum string literal length.

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.

C., Y. A. (2001), 'Multipulses of nonlinearly-coupled Schrödinger equations', *JOURNAL of Differential Equations* **173**(1), 92–137.

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. (1986*a*), 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. (1986*b*), 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., 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. (1991*a*), '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. (1991*b*), 'Numerical analysis and control of bifurcation problems: (II) Bifurcation in infinite dimensions', *Int. J. Bifurcation and Chaos* **1**(4), 745–772.

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., Knedlik, 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. (2001), Homoclinic branch switching: a numerical implementation of Lin's method, `http://www.enm.bris.ac.uk/research/reports/2001r11.ps.gz`, Applied Nonlinear Mathematics Research Report 2001.11, University of Bristol; *accepted by Int. J. Bifurcation and Chaos.*

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. (1995*a*), Constructing dynamical systems possessing homoclinic bifurcation points of codimension two, In preparation.

Sandstede, B. (1995*b*), Convergence estimates for the numerical approximation of homoclinic solutions, In preparation.

Sandstede, B. (1995*c*), 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.

Taylor, M. A. & Kevrekidis, I. G. (1989), Interactive AUTO : A graphical interface for AUTO86, Technical report, Department of Chemical Engineering, Princeton University.

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.