

WAVETRAIN

Software for Studying Periodic Travelling Wave
Solutions of Partial Differential Equations

JONATHAN A. SHERRATT

Dept of Mathematics and Maxwell Institute for Mathematical Sciences,
Heriot-Watt University, Edinburgh EH14 4AS, UK.

If you publish research done using this software, please cite

J.A. Sherratt: Numerical continuation methods for studying periodic travelling wave (wavetrain) solutions of partial differential equations. *Applied Mathematics & Computation* 218, 4684-4694 (2012).

If your research includes results on wave stability, please also cite

J.D.M. Rademacher, B. Sandstede, A. Scheel: Computing absolute and essential spectra using continuation. *Physica D* 229, 166-183 (2007).

and

J.A. Sherratt: Numerical continuation of boundaries in parameter space between stable and unstable periodic travelling wave (wavetrain) solutions of partial differential equations. *Advances in Computational Mathematics* 39, 175-192 (2013).

This user guide is for version wavetrain1.1

Copyright © 2011, 2013 by Heriot-Watt University, Edinburgh, UK. All rights reserved worldwide. Printing of parts or the whole of this document for the purposes of research or private study, or criticism or review, are permitted. No part of this document or the related files may be distributed by any means (electronic, photocopying, recording, or otherwise) without the prior written permission of either Heriot-Watt University or the author. All requests for such permission should be sent by e-mail to j.a.sherratt@hw.ac.uk; please include the word wavetrain (upper or lower case) in the subject heading.

Limit of Liability and Disclaimer of Warranty: The information provided in this document is provided “as is”. Heriot-Watt University and the author make no representation or warranties with respect to the accuracy or completeness of the contents of this document and specifically disclaim any implied warranties of merchantability or fitness for any particular purpose. Under no circumstances will Heriot-Watt University or the author be liable for any loss of use, interruption of business or any direct, indirect, special, incidental, and/or consequential damages of any kind (including loss of profits) even if Heriot-Watt University or the author have been advised of the possibility of such damages. Neither Heriot-Watt University nor the author warrant that this document will be error-free.

Contents

1	Introduction to WAVETRAIN and Installation	7
1.1	Aims and Scope	7
1.2	System Requirements	8
1.3	Authorship and Use of Other Software	9
1.4	Copyright, Distribution and Disclaimer	9
1.5	Acknowledgements	10
1.6	Feedback	10
1.7	Installation	11
2	How to Use WAVETRAIN	13
2.1	Getting Started With WAVETRAIN	13
2.1.1	Investigation of Periodic Travelling Wave Existence	13
2.1.2	Investigation of Periodic Travelling Wave Stability	21
2.2	Details of the WAVETRAIN Input Files	32
2.2.1	Details of <code>constants.input</code>	32
2.2.2	Details of <code>equations.input</code>	38
2.2.3	Details of <code>other_parameters.input</code>	45
2.2.4	Details of <code>parameter_list.input</code>	45
2.2.5	Details of <code>parameter_range.input</code>	45
2.2.6	Details of <code>variables.input</code>	46
2.3	A Worked Example	46
2.3.1	Stage 1 (Input): Create the Initial Input Files	47
2.3.2	Stage 2 (Run): Formulate a Wave Search Method	51
2.3.3	Stage 3 (Input): Add a Preliminary Wave Search Method	52
2.3.4	Stage 4 (Run): Look for Waves near the Hopf Bifurcation Locus	53
2.3.5	Stage 5 (Input): Amend the Wave Search Method	53
2.3.6	Stage 6 (Run): Test Runs of the <code>ptw</code> Command	54
2.3.7	Stage 7 (Input): Alter the File <code>constants.input</code>	55
2.3.8	Stage 8 (Run): Further Test Runs of <code>ptw</code> , and a Run of <code>ptw_loop</code>	55
2.3.9	Stage 9 (Run): Choose <code>nmesh2</code> for Stability Calculations	56
2.3.10	Stage 10 (Input): Change <code>order</code> in <code>constants.input</code>	58
2.3.11	Stage 11 (Run): Further Tests of Eigenvalue Convergence	58

2.3.12	Stage 12 (Input): Change <code>nmesh1</code> and <code>nmesh2</code> in <code>constants.input</code>	61
2.3.13	Stage 13 (Run): Test Runs of the <code>stability</code> Command	61
2.3.14	Stage 14 (Input): Change <code>nmesh3</code> in <code>constants.input</code>	61
2.3.15	Stage 15 (Run): Further Test Runs of <code>stability</code> , and a Run of <code>stability_loop</code>	61
2.3.16	Stage 16 (Input): Change <code>nmesh1</code> and <code>nmesh2</code> in <code>constants.input</code>	64
2.3.17	Stage 17 (Run): Run of <code>stability</code> for $\delta = -2$, $c = 3$	64
2.3.18	Stage 18 (Input): Reset <code>nmesh1</code> and <code>nmesh2</code> in <code>constants.input</code>	64
2.3.19	Stage 19 (Run): Plot the Results of the <code>stability_loop</code> Run	65
2.3.20	Stage 20 (Run): Run of <code>stability_boundary</code>	65
2.3.21	Stage 21 (Input): Reduce the Step Sizes for <code>stability_boundary</code>	66
2.3.22	Stage 22 (Run): Rerun of <code>stability_boundary</code>	66
2.3.23	Stage 23 (Run): Calculate Contours of Constant Period	67
2.3.24	Stage 24 (Input): Copy PDE Simulation Data File	68
2.3.25	Stage 25 (Run): Final Plots	69
2.3.26	Stage 26 (Run): Delete Unwanted Output Files	71
2.4	Documentation and Help Facilities	71
2.5	Troubleshooting	73
3	Advanced Features of WAVETRAIN	74
3.1	Further Details of Run Commands	74
3.1.1	The Format of Numerical Inputs, and Precision	74
3.1.2	Optional Arguments of the <code>stability_boundary</code> Command	75
3.1.3	Optional Arguments of the <code>period_contour</code> Command	75
3.1.4	The Commands <code>set_homoclinic</code> and <code>unset_homoclinic</code>	76
3.1.5	The Command <code>new_pcode</code>	78
3.1.6	The Command <code>bifurcation_diagram</code>	78
3.1.7	Keeping Track of WAVETRAIN Runs	83
3.1.8	The Parameter <code>iposim</code>	83
3.1.9	The Parameter <code>iwave</code>	86
3.1.10	Tracing the Loci of Folds	91
3.1.11	Commands for Deleting Data Files	94
3.1.12	Setting the Wave Search Method and Hopf Bifurcation Search Range in Complicated Cases	96
3.1.13	Details of the Matrix Eigenvalue Calculation	103
3.1.14	An Outline of How the <code>stability_boundary</code> Command Works	106
3.1.15	Multiple Simultaneous Runs of WAVETRAIN	116
3.2	Details of <code>defaults.input</code>	116
3.3	Further Details of Plot Commands	123
3.3.1	Changing Plotter Styles	123
3.3.2	Changing Other Plotter Settings	123

3.3.3	Labelling of Contours of Constant Period	125
3.3.4	Adding Additional Lines, Points and Text to Plots	125
3.3.5	Shading Regions of the Control Parameter–Wave Speed Plane . . .	128
3.3.6	Record Keeping for Postscript Plots	129
3.3.7	The Bounding Box for Postscript Plots	130
3.3.8	Abbreviations for Plot Commands	131
3.4	Details of Plotter Input Files	131
3.4.1	Details of <code>plot_defaults.input</code>	131
3.4.2	Details of Plotter Style File	139
4	Reference Guide to WAVETRAIN	146
4.1	Alphabetical List of Run Commands	146
4.2	Alphabetical List of Plot Commands	154
4.3	WAVETRAIN Directory Structure	158
4.3.1	The Structure of Subdirectories of the Main WAVETRAIN Directory	158
4.4	Details of Output Data Files	160
4.4.1	The Directory Structure of WAVETRAIN Output	161
4.4.2	pcode Subdirectories	162
4.4.3	bcode Subdirectories	164
4.4.4	ccode Subdirectories	165
4.4.5	ecode Subdirectories	167
4.4.6	fcode Subdirectories	168
4.4.7	hcode Subdirectories	168
4.4.8	rcode Subdirectories	169
4.4.9	scode Subdirectories	171
	Appendix: Commands Used to Generate Figures	173
	References	179

How to Use this User Guide

This user guide is rather long. However, new users should be reassured that only a relatively small proportion of it is needed in order to begin using WAVETRAIN. Part 1 of the user guide contains some introductory comments, followed by details of the installation procedure and of some post-installation tests. Part 2 is an introductory description of the package, via two examples, that explain how to use WAVETRAIN. Part 3 is concerned with more advanced features, and is intended for users who already have some familiarity with the package. Part 4 is a reference section, containing lists of commands and details of output data files.

In order to facilitate the printing of individual parts of this user guide, page numbering begins with the title page being page 1. This means that page selections in the print window give user guide pages with the same numbers.

Restrictions on Distribution

The WAVETRAIN software and this user guide are both freely available. However, please note that the formal statements on page 2 and in §1.4 specify that neither the user guide nor the software (in whole or in part) can be distributed in either electronic or printed (or any other) form. In particular, you cannot post this user guide or the information it contains, or part or all of the WAVETRAIN software, on any electronic bulletin board, web site, FTP site, newsgroup, or similar forum. The only places from which either the WAVETRAIN software or this document should be available are the WAVETRAIN web site www.ma.hw.ac.uk/wavetrain and the authors' web site www.ma.hw.ac.uk/~jas. Please respect this restriction: one of the reasons for it is to ensure that new users always download the most up-to-date version of the software and user guide.

Part 1

Introduction to WAVETRAIN and Installation

1.1 Aims and Scope

WAVETRAIN is a software package for investigating periodic travelling wave solutions of partial differential equations. Typically such equations will contain a number of parameters. WAVETRAIN requires that one of these be selected; this parameter will be termed the control parameter throughout this user guide. Focussing on periodic travelling waves introduces an additional parameter, the wave speed. The basic task performed by WAVETRAIN is the calculation of the region of the control parameter–wave speed plane in which periodic travelling waves exist, and additionally where they are stable.

WAVETRAIN is suitable for systems of equations with only single time derivatives, which can be coupled with equations involving space derivatives only. Mixed space and time derivatives are not allowed. Specifically, WAVETRAIN can be used to study systems of $N \geq 1$ equations whose form is either

$$\partial u_i / \partial t = F_i(\underline{u}, \partial \underline{u} / \partial x, \partial^2 \underline{u} / \partial x^2, \partial^3 \underline{u} / \partial x^3, \dots), \quad 1 \leq i \leq N \quad (1.1a)$$

or

$$\partial u_i / \partial t = F_i(\underline{u}, \partial \underline{u} / \partial x, \partial^2 \underline{u} / \partial x^2, \partial^3 \underline{u} / \partial x^3, \dots), \quad 1 \leq i < M \quad (1.1b)$$

$$0 = F_i(\underline{u}, \partial \underline{u} / \partial x, \partial^2 \underline{u} / \partial x^2, \partial^3 \underline{u} / \partial x^3, \dots), \quad M \leq i \leq N \quad (1.1c)$$

where $2 \leq M \leq N$. There is no upper limit on N or on the order of spatial derivatives on the right hand side of the equations.[†]

The three main underlying principles of WAVETRAIN are that it should

- (i) be easy to use, even for users with limited mathematical background;
- (ii) produce publication-quality graphical output that can be fine-tuned by the user if required;

[†]In fact, there is an upper limit on N : some of the temporary files created during WAVETRAIN runs are formatted under the assumption that $N < 2 \times 10^6$. This is not expected to be a limitation in any practical situation.

- (iii) provide detailed output data files that are accessible to the user if required, and that have an easily comprehensible format.

Once the user has acquired some familiarity with WAVETRAIN, application to a new problem should require a relatively small amount of user time. However, users should be aware that some of the computations may involve significant computer time; such runs can be performed in the background.

1.2 System Requirements

WAVETRAIN requires a unix-like operating system; thus linux and Mac OS X are both suitable. Users with Windows operating systems can use WAVETRAIN via a virtual machine, for which there is freely available software; see for example www.virtualbox.org/wiki. After downloading a virtual machine, the user will have to download a linux operating system, such as Ubuntu linux (www.ubuntu.com/download/ubuntu/download), which is freely available. Once a virtual machine and linux operating system have been downloaded, one simply opens the virtual machine and follows the installation procedure. For example, opening the Oracle VM VirtualBox gives the options **New**, **Settings**, **Start** and **Discard**. Selecting “New” opens a Virtual machine wizard window which guides the user through the setup of a new operating system; additional help is available from the VirtualBox user guide (www.virtualbox.org/wiki/Documentation).

Systems on which WAVETRAIN has been tested include an HP Compaq PC with the CentOS 6.4 (linux) operating system, a AMD64 PC running Oracle VM VirtualBox with the ubuntu-11.04-desktop-i386 operating system, and a MacBook Air with version 10.7.5 of the Mac OS X operating system.

WAVETRAIN also requires a Fortran77 compiler that is compatible with some extensions to the Fortran77 standard. Details of how to check whether your system has a suitable compiler are given in §1.7; if it does not, a number of suitable compilers are freely available. For example, WAVETRAIN has been tested with the GNU compiler gfortran-4.0, downloaded from the website packages.ubuntu.com/dapper/gfortran.

Graphical output in WAVETRAIN can be performed using either **gnuplot** or **sm**. The plotting commands are the same for both plotting packages. The **gnuplot** package is supplied automatically on many unix-like operating systems, but if it needs to be installed, it is freely available from www.gnuplot.info. The **sm** package is commercial but inexpensive, and can be obtained from www.supermongo.net. WAVETRAIN graphics were originally developed using **sm**, but it is anticipated that most users will use **gnuplot**. The plots generated by the two plotters are completely equivalent. The figures in this user guide were generated using **sm**.

To check whether **gnuplot** is installed on your system, simply type **gnuplot** at the system prompt. Either an error message will appear indicating that **gnuplot** is not installed, or **gnuplot** will start. In the latter case, type

```
show version long
```

at the **gnuplot>** prompt. This will give details of the version of **gnuplot** and various details of the compilation. WAVETRAIN requires version 4.2 or higher of **gnuplot**, and

requires that it has been compiled with the following options enabled: macro expansion, string variables, and data strings. These options will be indicated by “+MACROS”, “+STRINGVARS” and “+DATASTRINGS” being listed amongst the compile options. If one of these criteria is not met, then it will be necessary to reinstall or recompile `gnuplot` before it can be used as the plotter for WAVETRAIN. By default, WAVETRAIN uses `x11` as the `gnuplot` plotting terminal. To confirm that this is available on your system, type

```
set terminal x11 0 enhanced font "Sans,10"
```

at the `gnuplot>` prompt. If an error message appears, it will be necessary to edit the file `defaults.input` in order to change the terminal setting; see §3.4.1.2 for details.

To exit `gnuplot`, type `quit` at the `gnuplot>` prompt.

1.3 Authorship and Use of Other Software

WAVETRAIN was written by Jonathan Sherratt. It includes (with permission) software from several other packages:

AUTO97, developed by Eusebius Doedel (currently Concordia University, formerly California Institute of Technology) and coworkers (indy.cs.concordia.ca/auto; Doedel, 1981; Doedel *et al.*, 1991). AUTO97 is a numerical continuation package that is used extensively by WAVETRAIN. Current AUTO users should note that the relevant parts of the AUTO code are incorporated into WAVETRAIN, which will therefore run independently of existing AUTO settings. Note that previous experience with AUTO is not necessary for using WAVETRAIN. The AUTO97 distribution includes code from the software package EISPACK, a software package for numerical computation of eigenvalues and eigenvectors of matrices, and this code is also distributed as part of WAVETRAIN. EISPACK was developed by Brian Smith and coworkers at Argonne National Laboratory (www.netlib.org/eispack; Smith *et al.*, 1976).

LAPACK, a software package for numerical linear algebra developed by researchers at many institutions (www.netlib.org/lapack; Anderson *et al.*, 1999). Note that the EISPACK package used by AUTO97 is a forerunner of LAPACK.

BLAS, the Basic Linear Algebra Subprograms, developed by Jack Dongarra (currently University of Tennessee, formally Argonne National Laboratory) and coworkers (www.netlib.org/blas; Lawson *et al.*, 1979; Dongarra *et al.*, 1988a,b; Dongarra *et al.*, 1990a,b).

MINPACK, a software package for studying nonlinear systems of algebraic equations, developed by the University of Chicago, as operator of Argonne National Laboratory (www.netlib.org/minpack; Moré *et al.*, 1984).

1.4 Copyright, Distribution and Disclaimer

Copyright. WAVETRAIN software is intended for free personal private and academic use only and was developed by Jonathan Sherratt at Heriot-Watt University, Edin-

burgh, UK, and is protected by copyright. The copyright owner is Heriot-Watt University, with the exception of the routines from the packages AUTO, BLAS, EISPACK, LAPACK and MINPACK, which are included with permission, which are clearly identified in the source code, and the use of these packages is under the conditions of the original owners which are available at the websites www.netlib.org/blas/faq.html#2; www.netlib.org/lapack/#_licensing; www.netlib.org/minpack/disclaimer.

Persons or organisations wishing to use this WAVETRAIN software for commercial purposes or purposes other than its original intention should contact the owners to request specific written permissions.

Distribution. Distribution of WAVETRAIN software, in part or as a whole, by any means (electronic, printed copy or otherwise), is prohibited without the prior written permission of either Heriot-Watt University or the author Jonathan A. Sherratt. All requests for such permission should be sent by e-mail to j.a.sherratt@hw.ac.uk; please include the word wavetrain (upper or lower case) in the subject heading.

Warranty limitations. The product and service are provided “as is”. The owner disclaims any and all warranties, including but not limited to, all expressed or implied warranties of merchantability and fitness for a particular purpose. Under no circumstances will the owner be liable to you for any loss of use, interruption of business or any direct, indirect, special, incidental, and/or consequential damages of any kind (including loss of profits) even if the owner has been advised of the possibility of such damages. The owner accepts no legal liability for any of the product and/or service sold, offered for sale, or otherwise made available for public or private consumption in relation to this software. The owner does not warrant that this software will be error-free.

1.5 Acknowledgements

While writing version 0.0 of WAVETRAIN, I was supported in part by a Leverhulme Trust Research Fellowship. I would have been incapable of writing WAVETRAIN without my research collaboration with Matthew Smith (Microsoft Research, Cambridge) and Jens Rademacher (CWI, Amsterdam), and I am very grateful to them both for our many discussions. I also thank Philipp Janert and Wasit Limprasert for help with gnuplot, Steve Mowbray for computer advice, Lisa Keyse for secretarial help, and Ayawoa Dagbovie, Osman Gani, Alan MacDonald and Jennifer Reynolds for testing the code and parts of this user guide. Finally but most importantly, I am deeply grateful to Pam and Dorothy for being so supportive, and for putting up with my many hours in front of the computer screen.

1.6 Feedback

Feedback about WAVETRAIN is strongly encouraged. It is particularly helpful to receive details of any problems experienced in using WAVETRAIN, and any suggestions for improvements that could be included in future releases. Comments about this user guide, including typos, are also very welcome. Positive comments are also permitted! The e-mail address for all feedback is j.a.sherratt@hw.ac.uk; please include the word wavetrain

(upper or lower case) in the subject heading. Please state in your e-mail which version of WAVETRAIN you are using; this information is contained in the file `README` in the main WAVETRAIN directory.

1.7 Installation

WAVETRAIN is freely available and can be downloaded from the WAVETRAIN web site www.ma.hw.ac.uk/wavetrain. The single file `wavetrainn.m.tar` should be downloaded; here *n.m* is the version number. This file should be expanded in the usual way, by typing

```
tar -xf wavetrainn.m.tar
```

at the system prompt. This will create a directory `wavetrainn.m` containing all of the WAVETRAIN files. The directory `wavetrainn.m` will be referred to as “the main WAVETRAIN directory” throughout this user guide. The tar file can then be deleted by typing

```
rm -f wavetrainn.m.tar
```

although this is not necessary. Now go into the new direcorey by typing

```
cd wavetrainn.m
```

and then type

```
./install
```

which will list the copyright, distribution and disclaimer statement. Note that the “./” before the command is recommended because the search path might not include the current directory when this command is run; this issue is discussed further below. The user must enter `y` or `Y` at the prompt to confirm that they have read and understood the statement, and that they agree to the conditions.

After completing the installation, it is recommended that the user runs three tests:

Test 1: the search path. WAVETRAIN requires that the current directory (“.”) is included in the search path. A user already using unix will almost certainly have this setting, but this may not be the case for users who are new to unix. To check this setting, the command

```
./pathtest
```

should be entered at the system prompt (from within the main WAVETRAIN directory). This will check the search path, and if a change is required, the screen output will contain advice on how to change the search path.

Test 2: the Fortran77 compiler. Once the search path is set appropriately, the user should run a test of the Fortran77 compiler on their system. The default name for the compiler is `f77`, but this can be changed by editing the file `defaults.input` in the directory `input_files`; the compiler name is set on line 15 (see §3.2 for details). To test the compiler, one types the command

fortrantest

in the main WAVETRAIN directory. If the resulting screen output is

Test completed

then the compiler is suitable for WAVETRAIN. If an error message appears, such as “f77: Command not found”, this indicates that a compiler needs to be installed, or that the name needs to be changed: a number of compilers are freely available. A third possibility is that “Test completed” appears, but is preceded by a list of subroutines, such as

MAIN:
dotest:

which indicates that by default, the compiler operates in a non-silent mode. This type of compiler output will make WAVETRAIN’s screen output very difficult to follow, and it should be suppressed. This can be done by editing **defaults.input** in the **input_files** subdirectory (see §3.2) and adding a suitable compiler flag on line 15. The appropriate flag is compiler dependent, but “-silent” or “-fsilent” are likely possibilities. Having made this change, the **fortrantest** command should be run again to confirm that it has worked.

Test 3: a test run of AUTO97. WAVETRAIN makes extensive use of the numerical continuation software AUTO97 (see §1.3). Having confirmed that the Fortran77 compiler is working appropriately, the user is recommended to test the operation of AUTO97, via the command

auto97test

(from within the main WAVETRAIN directory). This test will begin by compiling the AUTO97 code. This may not be successful even if **fortrantest** has worked, because AUTO97 uses some extensions to the Fortran77 standard. WAVETRAIN requires that the user’s Fortran77 compiler is able to compile the AUTO97 code. If the compilation is successful, the **auto97test** command goes on to perform a test run of AUTO97, and the output from this will be compared with reference output. If the two sets of output are either identical or have only small numerical differences, then the test will be declared successful.

Once these three tests have been completed successfully, the user is ready to begin using WAVETRAIN.

Part 2

How to Use WAVETRAIN

2.1 Getting Started With WAVETRAIN

The main WAVETRAIN directory contains a subdirectory `input_files`. Before starting investigation of a new periodic travelling wave problem, the user must create a subdirectory of `input_files` in which the various input files must be placed. Several such input subdirectories are provided in the installed version of WAVETRAIN, and the input subdirectory `demo` contains the input files for a very simple application of WAVETRAIN, intended as a tutorial case. The equations studied in `demo` are

$$\begin{aligned}\partial u / \partial t &= wu^2 - Bu + \partial^2 u / \partial x^2 \\ \partial w / \partial t &= A - w - wu^2 + \nu \partial w / \partial x.\end{aligned}$$

This is a dimensionless version of a model for vegetation patterns in semi-arid environments, first proposed by Klausmeier (1999). For a more detailed discussion of this problem, and for details of a more systematic study of these equations using WAVETRAIN, see Sherratt (2012). A description of the various input files is postponed until §2.2, and the remainder of this section consists of an explanation of the use of WAVETRAIN to study the problem in `demo`. New users are encouraged to run the various commands on their own system as they follow the description of this example.

2.1.1 Investigation of Periodic Travelling Wave Existence

The typical starting point for the investigation of periodic travelling waves is a series of partial differential equation simulations in which such waves have been observed. Based on this (or other) information, the user must decide upon a range of values of the control parameter and wave speed over which periodic travelling waves are to be investigated; this information is entered in the input file `parameter_range.input`. For `demo` the range $0.1 < A < 3.4$ and $0.3 < c < 1.1$ has been chosen. Here A has been chosen as the control parameter, and c is the name chosen for the wave speed; these are set in the input file `variables.input`. The first stage in the use of WAVETRAIN is to test the suitability of (some of) the various computational inputs in the file `constants.input` for the particular problem. To do this, one must enter manually a pair of A and c values for which a periodic travelling wave is expected to exist: a good choice is values corresponding

```
This run has been assigned run code 1001
STARTING CALCULATION OF PERIODIC TRAVELLING WAVE SOLUTION

THE AUTO LIBRARIES NEED TO BE CREATED

THE AUTO SOURCE CODE WILL NOW BE RECOMPILED

END OF CALCULATION FOR demo WITH pcode=100 rcode=1001

THE OUTCOME WAS:
    PERIODIC TRAVELLING WAVE FOUND, STABILITY NOT REQUESTED
    PERIOD = 0.21580722E+02

THERE WERE NO ERRORS OR WARNINGS DURING THE RUN
```

Figure 2.1: The output that should appear on the screen during the execution of the command `ptw demo 2.0 1.0`. There may also be a message about warnings generated during the compilation of the AUTO source code. The precise value of the period may vary according to the computer architecture and Fortran77 compiler.

to the observation of a wave in simulations of the partial differential equations. For `demo`, the values $A = 2.0$ and $c = 1.0$ are suitable. One then types

```
ptw demo 2.0 1.0
```

and WAVETRAIN will calculate the corresponding periodic travelling wave. Figure 2.1 shows the screen output that should appear during the run, which should only take a few seconds. The run is allocated a four digit “rcode” (1001-9999) which will be used later to specify the run, for example when plotting. Since this is the first run, the rcode value is 1001, and this is reported in the screen output. One can then calculate the wave for a second pair of control parameter and wave speed values, for example via the command

```
ptw demo 1.5 0.8
```

which is allocated an rcode value of 1002. Often it will be appropriate to run tests for several pairs of values.

The user may wish to plot the periodic travelling waves that have been calculated. To do this, one starts the WAVETRAIN plotter by typing

```
plot demo
```

and one then types

```
@ptw 1001
```

at the plotter prompt to obtain a plot of the first wave that was calculated, and

```
@ptw 1002
```

for the second. The results are illustrated in Figure 2.2. The plotter should then be exited by typing `@quit` or `@exit`. Note that all plotter commands begin with the “at” character `@`.

Having confirmed that the computational constants are appropriate one can proceed with investigation of the designated section of the A – c parameter plane. The input file `parameter_range.input` also contains the number of points in a grid of (A, c) parameter values over which wave existence will be checked. For `demo`, a 5×3 grid is used. This is extremely coarse, and is chosen to reduce run times for demonstration purposes; typically a 10×10 or 20×20 grid would be used in a real problem. To run over the grid, one types

```
ptw_loop demo
```

and WAVETRAIN will loop through the points in the parameter grid. As will be seen in the screen output, WAVETRAIN allocates a 3 digit “pcode” value to this run (the “p” in pcode stands for “parameter loop”). Apart from rcode values (which are four-digit), all WAVETRAIN code numbers, including pcode values, are three-digit and range from 101 to 999. Since this is the first run to be performed, the pcode value will be 101.

To visualise the results of the run, one starts the WAVETRAIN plotter again by typing

```
plot demo 101
```

(the number 101 is the pcode value) and then types

```
@pplane
```

at the plotter prompt. The plotter asks whether or not to use the default setting of labelling the parameter sets by their rcode values. Hitting RETURN (ENTER) to accept this default gives the picture illustrated in Figure 2.3a: the colour of an rcode value indicates the result of the corresponding calculation. Note that a rather small font is used for the rcode values in Figure 2.3a. This default setting is chosen with much finer parameter grids in mind (see for example Figure 3.7 on page 90). However, like most aspects of WAVETRAIN plots, the default font can be changed (see §3.3.1).

A key indicating the meaning of the colours is automatically generated when the plotter is started; it is in the file `pplane_key.eps` in the subdirectory `postscript_files` of the main WAVETRAIN directory. The colours can all be changed if desired (see §3.3.1), but with their default values, the key will be as shown in Figure 2.4. Figure 2.3a shows that there are two different outcomes, depending on the control parameter and wave speed values. For some pairs of values, a periodic travelling wave has been found, while for others it has not. In the latter case, the phrase “no convergence” appears beside the corresponding colour in the key in Figure 2.4. To search for periodic travelling waves, WAVETRAIN numerically continues periodic travelling waves along the solution branch as the control parameter varies, starting at the Hopf bifurcation point in the travelling wave equations. The phrase “no convergence” indicates that this continuation has ended with a convergence failure because the solution approaches a homoclinic solution (the end of

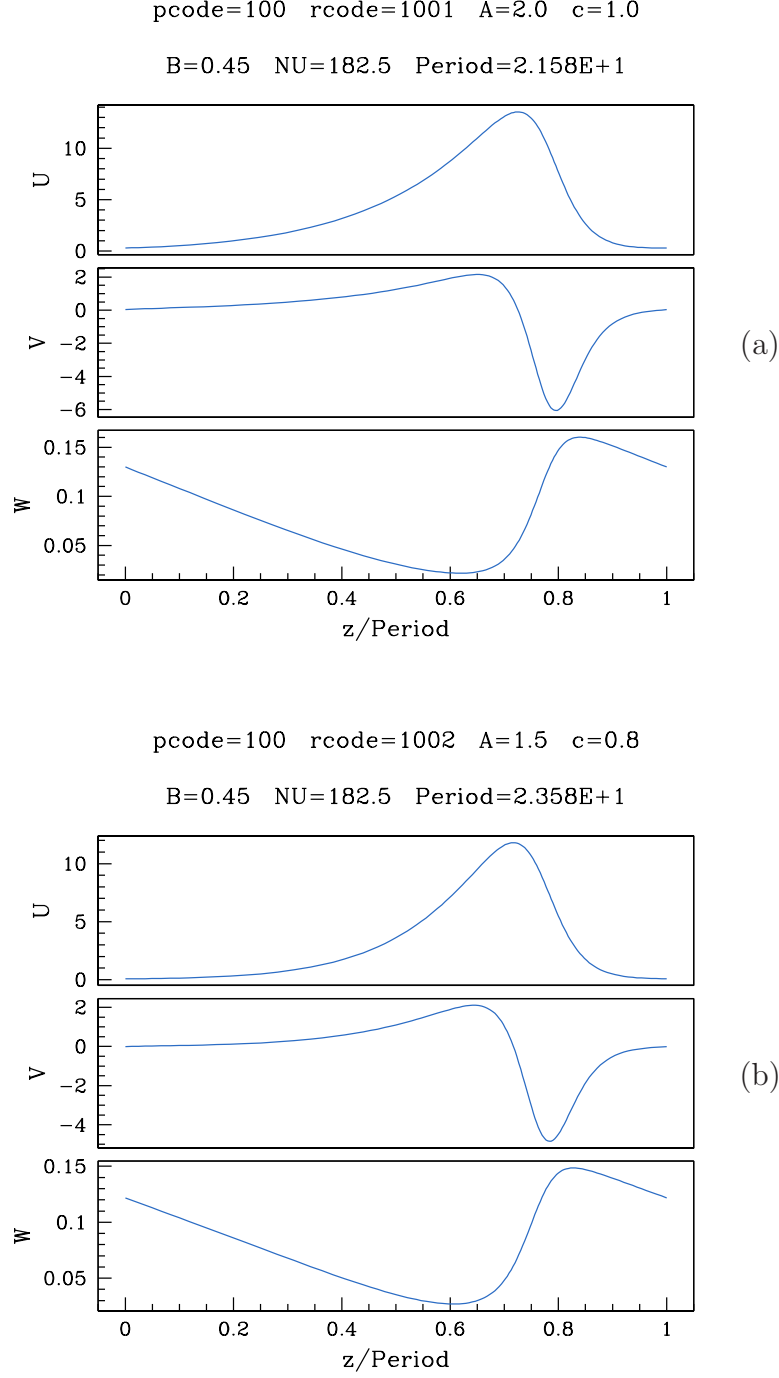


Figure 2.2: Two periodic travelling wave solutions for the problem `demo`. The run and plot commands used to generate these figures are given in the main text, and are also listed in the Appendix. The first lines of the titles show the pcode and rcode numbers for the run, together with the values of the control parameter A and the wave speed c . The second lines show the periods of the solutions, and also the values of the two parameters B and ν , which appear in the equations but which are not being used as the control parameter; their values are specified in the file `other_parameters.input` (see §2.2.3).

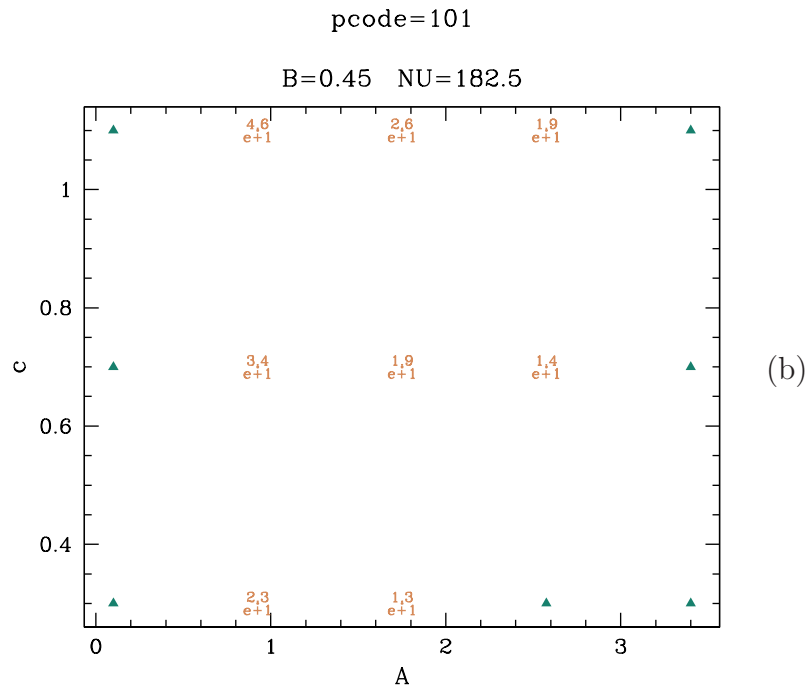
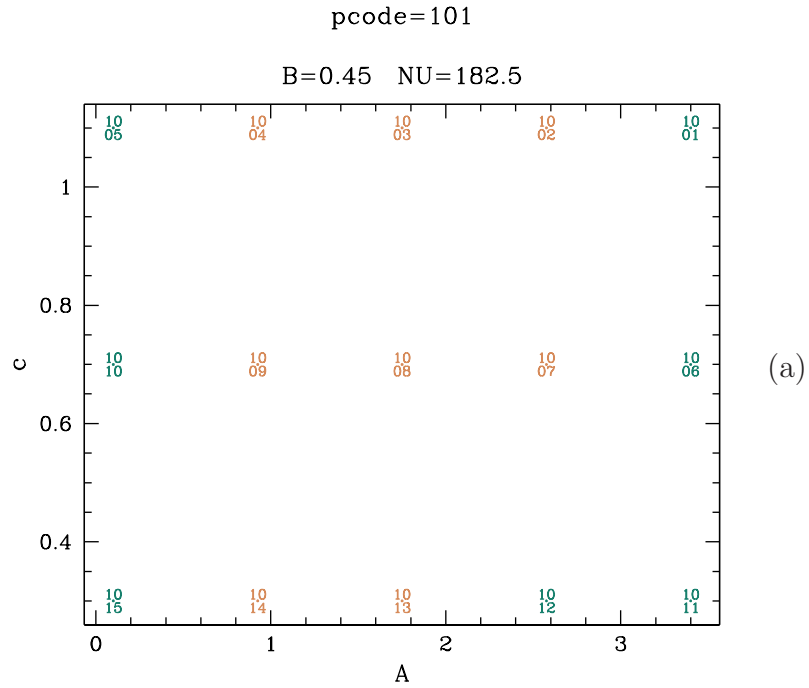


Figure 2.3: Results on the existence of periodic travelling wave solutions for the problem **demo**. A key illustrating the meaning of the various colours is shown in Figure 2.4 on page 18. The run and plot commands used to generate these figures are given in the main text, and are also listed in the Appendix.

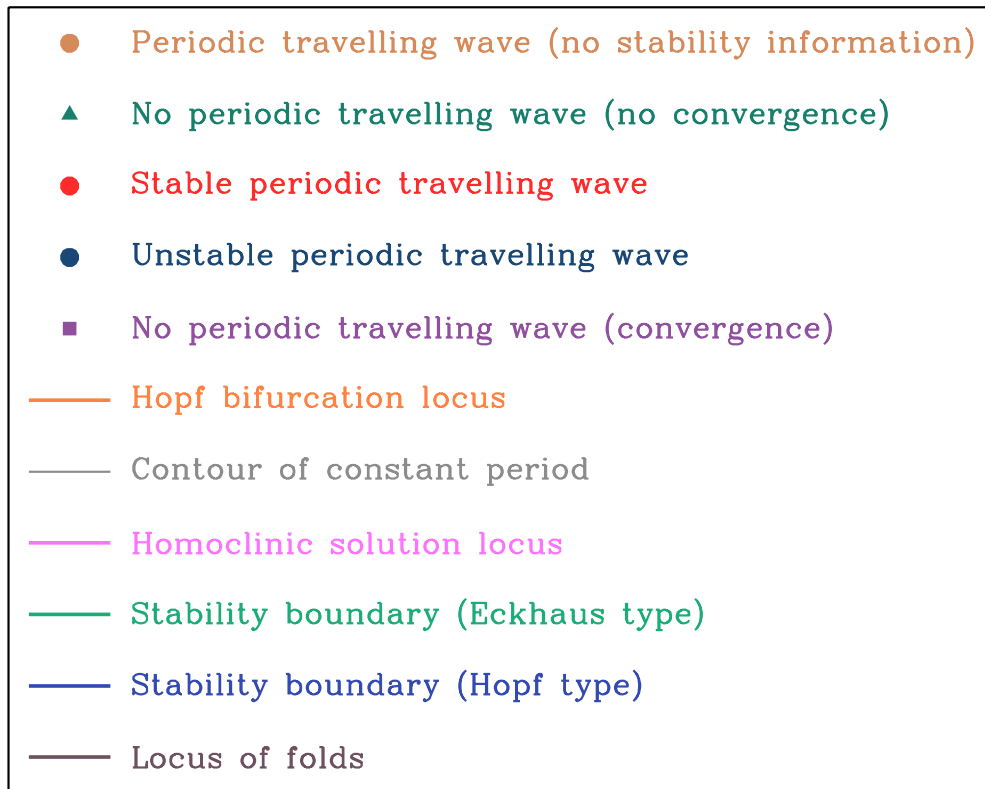


Figure 2.4: The key for the control parameter–wave speed plots, indicating the meaning of the various points and lines. This key is automatically generated when the plotter is started; it is in the file `pplane_key.eps` in the relevant subdirectory of `postscript_files` (e.g. `postscript_files/demo`); note that `postscript_files` is a subdirectory of the main WAVETRAIN directory. The user can specify that only a subset of the lines in this key be included: see §3.3.1 for details on how to change this and other plotter settings. Additionally, if the user wishes to include a key in material for presentation or publication, then they will probably prefer to use a single entry for cases where there is no periodic travelling wave, and correspondingly to have a single entry in the key. This can be achieved by resetting the plotter setting `pplanekeytype` (see §3.3.1).

the solution branch) without passing through the required value of the control parameter. There are some situations in which there is no periodic travelling wave but for which continuation finishes without a convergence failure; the WAVETRAIN plotter then uses a different symbol. This issue is discussed in detail in §3.1.9. If the user wishes to include a key in material for presentation or publication, then they will probably prefer not to include the “no convergence” phrase, and this can be achieved by changing the plotter settings, as explained in §3.3.1.

The form of the periodic travelling wave solutions can again be visualised using the `@ptw` plotter command. Thus

```
@ptw 1007
```

would generate a plot of the wave for $A = 2.574$ and $c = 0.7$, while

```
@ptw 1006
```

would give an error message, since no periodic travelling wave solution was found for this rcode value. Rerunning the `@pplane` plot command but now entering `period` as the label style gives the picture shown in Figure 2.3b; the colours are the same as in the previous picture, but when the wave exists, its period is now displayed. The plotter should now be exited by typing `@quit` or `@exit`.

At this stage one could proceed with investigation of the stability of the periodic travelling wave solutions, but in this case we will first use WAVETRAIN to obtain more details of the region of the A - c parameter plane in which periodic travelling wave solutions exist. Boundaries of the region can be of two possible types. They can correspond to a Hopf bifurcation in the travelling wave equations, so that the wave amplitude is zero on the boundary. Alternatively, they can correspond to a homoclinic solution, so that the period (wavelength) is infinite. The parameter grid used for `demo` is too coarse to clarify the nature of the left- and right-hand boundaries in Figure 2.3. (The coarse grid is chosen to keep run times small). However with a finer grid, a plot of wave periods such as Figure 2.3b shows that the period increases markedly near the boundary on the left, but not near that on the right. Hence the left-hand boundary corresponds to homoclinic solutions, while the right-hand boundary corresponds to Hopf bifurcation points.

WAVETRAIN can be used to trace these boundaries. From the previous results, it is expected that the Hopf bifurcation locus (right hand boundary) crosses $c = 0.8$ between $A = 2.0$ and $A = 3.5$. Typing

```
hopf_locus demo 101 2.0 0.8 3.5 0.8
```

causes WAVETRAIN to locate this crossing point and then continue the boundary starting from that point. (The number 101 is the pcode value.) Similarly, the previous results suggest that the locus of homoclinic solutions (left hand boundary) crosses $c = 0.8$ between $A = 0$ and $A = 1.5$. There is no capability in WAVETRAIN to locate/track true homoclinic solutions but one can approximate the homoclinic (period= ∞) locus by the locus of a solution of large finite period. Typing

```
period_contour demo 101 period=3000 1.5 0.8 0.0 0.8
```

causes WAVETRAIN to calculate the contour of period 3000. WAVETRAIN allocates a 3 digit “ccode” number (101-999) to this period contour calculation; since this is the first such calculation to be done, the ccode number is 101. In many cases, one will wish to calculate contours for various different values of the period, and these will be given different code numbers. Thus typing

```
period_contour demo 101 period=80 2.5 0.8 0.5 0.8
```

causes WAVETRAIN to calculate the contour for waves of period 80; the ccode number for this run is 102. The command `list_periods` can be used to list the periods corresponding to the various ccode numbers; thus for instance

```
list_periods demo 101
```

(where 101 is the pcode value) causes WAVETRAIN to report that the period is 3000 for ccode 101 and 80 for ccode 102. An “hcode” number (101-999) is also allocated to a run of `hopf_locus`; this is because in principle there may be more than one Hopf bifurcation locus curve in the chosen part of the A - c plane; however this is not the case for `demo`. Note that the `period_contour` command also has optional additional arguments, which are discussed in §3.1.3.

To visualise these new results, one starts the plotter again by typing

```
plot demo 101
```

and then draws the parameter grid results again by typing

```
@pplane
```

at the plotter prompt; for variety, select `symbol` as the plotting type. One can then draw the Hopf bifurcation locus by typing

```
@hopf_locus 101
```

where 101 is the hcode number. If more than one Hopf bifurcation locus has been calculated, the command

```
@hopf_locus all
```

can be used to plot all of them. For the period contours, the separate commands

```
@period_contour 101
```

and

```
@period_contour 102
```

can be used to draw the contour for the two ccode numbers; alternatively the single command

```
@period_contour all
```

will draw them both. In either case, it is appropriate to press RETURN (ENTER) when prompted about the label location, to select default labelling. The plot resulting from these various commands is illustrated in Figure 2.5. The plotter should now be exited by typing `@quit` or `@exit`.

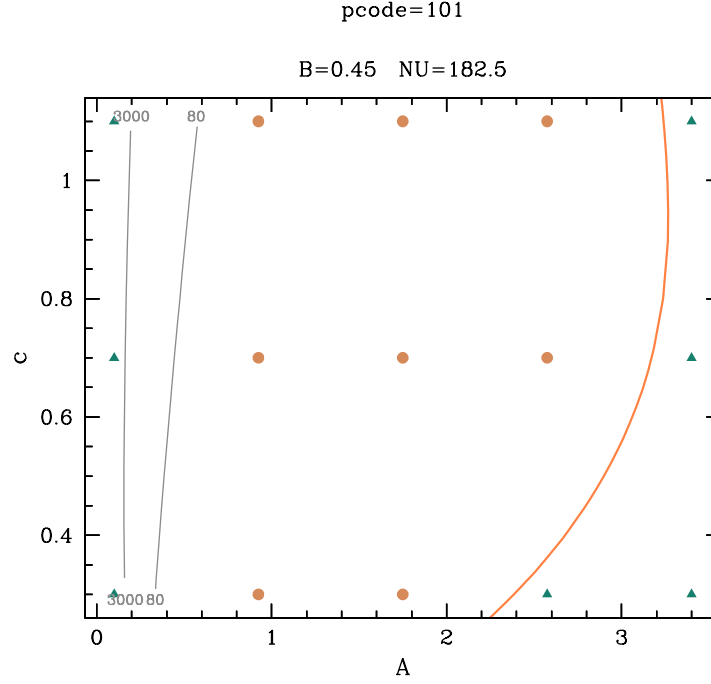


Figure 2.5: The Hopf bifurcation locus and two contours of constant wave period for the problem `demo`, superimposed on a plot showing periodic travelling wave existence in the control parameter–wave speed plane. A key illustrating the meaning of the various colours and symbols is shown in Figure 2.4 on page 18. The run and plot commands used to generate this figure are given in the main text, and are also listed in the Appendix.

2.1.2 Investigation of Periodic Travelling Wave Stability

The calculations and plots described above give a detailed account of the region of the A – c parameter plane in which there are periodic travelling waves, but they give no information about the stability of these waves, which is a fundamental issue in applications (Sherratt & Smith, 2008). WAVETRAIN determines stability by calculating the eigenvalue spectrum via numerical continuation, using the method of Rademacher *et al* (2007). Although a detailed understanding of the method is not needed in order to use WAVETRAIN, a brief introduction will help users with their choice of computational constants. For a periodic travelling wave, the system of equations satisfied by the (complex-valued) eigenfunctions has coefficients that are periodic in the travelling wave coordinate. However, the eigenfunctions themselves need not be periodic; their amplitude must be periodic, but their phase shift γ across one period of the wave is not constrained, and is the central player in the method of Rademacher *et al* (2007). One begins by calculating the eigenvalues for which $\gamma = 0$, i.e. those for which the corresponding eigenfunction is periodic with the same period as the wave. This is done by discretising the eigenfunction equation and then

solving (numerically) the resulting matrix eigenvalue problem. These periodic eigenfunctions and the corresponding eigenvalues can then be used as starting points for numerical continuation in the phase shift γ , which effectively “joins the dots” in the eigenvalue complex plane. Performing this continuation starting from each periodic eigenfunction results in the whole eigenvalue spectrum being traced out. This spectrum will always contain the origin (reflecting the neutral stability of the wave to translations), and the wave is unstable if and only if the spectrum crosses into the right-hand half of the eigenvalue complex plane.

In order to implement this procedure, WAVETRAIN requires two basic computational parameters to be set:

1. The number of periodic eigenfunctions to use as starting points for continuation in γ (**nevalues**). WAVETRAIN uses the eigenvalues with largest real part, considering either eigenvalues with real part ≥ 0 , or all eigenvalues, depending on the value of the constant **iposim** (see §3.1.8).
2. The number of points used in the discretisation of the eigenfunction equations (**nmesh2**).

In principle, the spectrum may undergo a large excursion that crosses the imaginary axis between any pair of consecutive eigenvalues corresponding to periodic eigenfunctions, but in practice it is typical that a relatively small value of **nevalues** (8–10 say) is sufficient to capture any crossing of the imaginary axis. For demo **nevalues** is fixed at 5 to reduce computation time; this would be dangerously low for a real problem.

The choice of **nmesh2** is very important. If it is too low, the discretisation introduces eigenvalues that have no analogue in the actual partial differential equations. However too high a value makes computation times excessive and also introduces significant errors into the matrix eigenvalue calculation (see §3.1.13). The command **eigenvalue_convergence** enables a suitable value of **nmesh2** to be determined. It has a variable (> 3) number of arguments. The first three arguments are the problem name (**demo** in this case) and the values of the control parameter and wave speed being considered; the rest are a list of candidate **nmesh2** values. The command **eigenvalue_convergence** sorts these candidate values into numerical order and calculates the requested number of eigenvalues (according to **nevalues** and **iposim**). Thus

```
eigenvalue_convergence demo 2.4 0.8 5 10 30 100 200
```

calculates the five eigenvalues for **nmesh2** = 5, 10, 30, 100 and 200. (Five is the value of **nevalues** set in the input file **constants.input**; see §2.2.1). The run is allocated an “ecode” value (101-999); since this is the first run of **eigenvalue_convergence**, the ecode value is 101. Note that this run takes several minutes on a typical desktop computer.

The **eigenvalue_convergence** command does not display any results itself. Rather the separate command **convergence_table** is used for that purpose. For example, one types

```
convergence_table demo 101 2
```

(here 101 is the ecode value) to display a “convergence table”, showing how the second out of the five eigenvalues changes as `nmesh2` is varied. Here the eigenvalues are ordered by the size of their real parts. The results are shown in Figure 2.6a, but note that the precise numerical values will vary according to computer architecture and the Fortran77 compiler. Alternatively one can use

```
convergence_table demo 101 3 4
```

to display a table using eigenvalues number 3 and 4 (ordered by their real parts). The results of this are shown in Figure 2.6b, but again note that the precise numerical values will vary according to computer architecture and the Fortran77 compiler. Finally one can use the command

```
convergence_table demo 101 all
```

to show all five of the eigenvalues (results not illustrated). The results in Figure 2.6b show that the eigenvalue ≈ -0.426 is missing in the (very coarse) discretisation at `nmesh2`=5, but is present at `nmesh2`=10; further increase improves accuracy. The choice `nmesh2`=30 is adequate for the purposes of demo. Note that as well as listing the real and imaginary parts of the eigenvalues, the “convergence table” produced by the “`convergence_table`” command contains an estimated error bound for each eigenvalue displayed. This error bound concerns the numerical errors in the calculation of the matrix eigenvalues. Therefore the displayed eigenvalues are only reliable as matrix eigenvalues if this error bound is significantly smaller in absolute value than both the real and imaginary parts of the calculated eigenvalue.

Having fixed on `nevalues`=5 and `nmesh2`=30, one can test these choices by calculating the eigenvalue spectrum. In practice, the convergence tests described above can only be done for one or at most a few pairs of control parameter and wave speed values, and one assumes that the appropriate values of `nevalues` and `nmesh2` are not sensitive to parameters. Thus test calculations need not be done for the same control parameter and wave speed values as those used in the convergence tests, and a suitable command would be

```
stability demo 2.2 0.8
```

where the second and third arguments are the values of A and c . This run will be given a (four-digit) rcode value, which will be reported in the screen output; since two runs (of the `ptw` command) have been done previously for a specified parameter set, the rcode value will be 1003 in this case. It can be visualised by typing

```
plot demo
```

to start the plotter; no pcode value is needed since the results to be plotted do not come from a parameter plane run. The command

```
@spectrum 1003
```

(a) Eigenvalue 2

nmesh	Re of eval 2	Im of eval 2	Error bound 2
5	-0.38268891E+00	0.76626544E-01	0.33022440E-12
10	-0.35918911E+00	0.00000000E+00	0.19255672E-11
30	-0.36096484E+00	0.00000000E+00	0.60103036E-11
100	-0.36100219E+00	0.00000000E+00	0.22931134E-10
200	-0.36102336E+00	0.00000000E+00	0.93608241E-10

(b) Eigenvalues 3 and 4

nmesh	Re of eval 3	Im of eval 3	Error bound 3	Re of eval 4	Im of eval 4	Error bound 4
5	-0.38268891E+00	-0.76626544E-01	0.33022440E-12	-0.50461854E+00	0.37706544E+00	0.11312082E-12
10	-0.42579712E+00	0.00000000E+00	0.18899599E-11	-0.51709584E+00	0.41066600E+00	0.28770486E-12
30	-0.42668430E+00	0.00000000E+00	0.73063981E-11	-0.52190354E+00	0.42118978E+00	0.61459815E-12
100	-0.42665738E+00	0.00000000E+00	0.28518558E-10	-0.52245312E+00	0.42252182E+00	0.25078373E-11
200	-0.42662685E+00	0.00000000E+00	0.11732264E-09	-0.52249548E+00	0.42262715E+00	0.10448786E-10

Figure 2.6: The change in the eigenvalues of the discretised eigenfunction equations as the constant `nmesh2` (defined in `constants.input`) is varied. The run command for these tables is `eigenvalue_convergence demo 2.4 0.8 5 10 30 100 200`. Assuming that this is the first run of the `eigenvalue_convergence` command for `demo`, it will be allocated an `ecode` value of 101. The tables shown can then be generated via the commands (a) `convergence_table demo 101 2`, (b) `convergence_table demo 101 3 4`. However, note that the precise numerical values will vary according to computer architecture and the Fortran77 compiler.

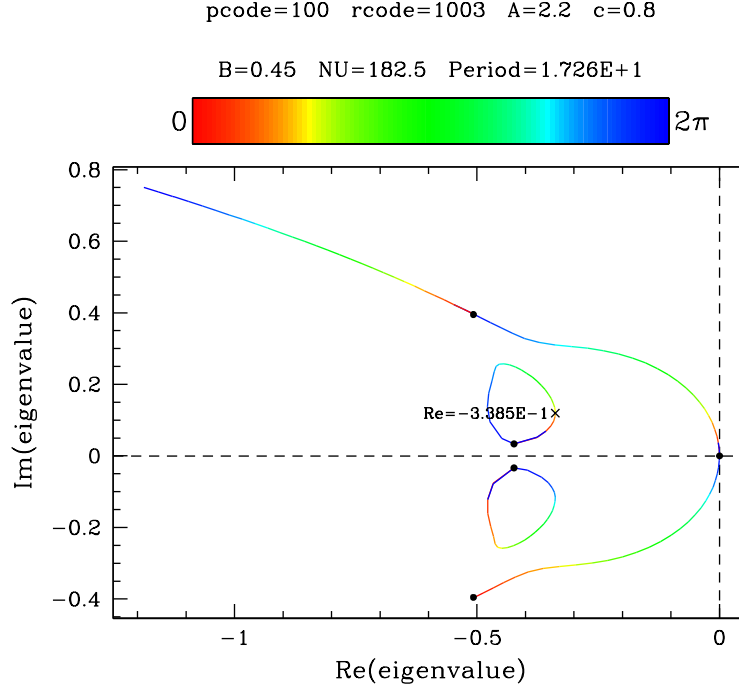


Figure 2.7: The eigenvalue spectrum for one pair of control parameter and wave speed values, for the problem `demo`. The dots indicate eigenvalues corresponding to periodic eigenfunctions, and the colours along the curve indicate the phase difference in the eigenfunction over one period of the periodic travelling wave. The run and plot commands used to generate this figure are given in the main text, and are also listed in the Appendix. The two parameters B and ν , whose values are given in the title, appear in the equations but are not being used as the control parameter. Their values are specified in the file `other_parameters.input` (see §2.2.3). The cross, which is labelled with the corresponding value of the real part of the eigenvalue, indicates the fold in the eigenvalue spectrum with largest real part, other than the origin. The sign of the real part of the eigenvalue at this point is used by `WAVETRAIN` to classify the wave as either stable or unstable. Note that if plotting is done using `gnuplot` then ticmarks will be absent from the horizontal axis in this plot; this applies if `spectrumcolour` is set to `rainbow` or `greyscale` or `grayscale`, but not if a single colour is used.

generates a plot of the eigenvalue spectrum (here 1003 is the `r`code value). When prompted about the axes limits, one can just press RETURN (ENTER), since the default settings are appropriate; if `gnuplot` is used as the plotter, there will also be a prompt about rescaling, and again one can just press RETURN (ENTER), since the default scaling is suitable. The resulting plot is illustrated in Figure 2.7; the dots indicate eigenvalues corresponding to periodic eigenfunctions ($\gamma = 0$), and the colours (red–green–blue) indicate the value of γ along the spectrum curves in between. Note that both the colour scheme for the spectrum curves and the marking of eigenvalues corresponding to periodic eigenfunctions are default settings that can be changed (see §3.3.2). As usual, one exits the plotter by typing `@quit` or `@exit`.

Having chosen values of `nevalues` and `nmesh2`, investigation of periodic travelling wave stability using `WAVETRAIN` is a two stage process. First, one scans the A – c parameter plane looking for regions in which periodic travelling waves are stable/unstable. This is done using the command `stability_loop`, which is directly analogous to `ptw_loop` except that at each parameter point at which a wave exists, its stability is also determined. Thus the command

```
stability_loop demo
```

loops through the points in the parameter plane as specified in `parameter_range.input`. This run will be allocated a new `p`code value; since only one previous parameter plane has been determined (using `ptw_loop`), the `p`code value is 102. Note that this run of `stability_loop` repeats the computations done previously in the run of `ptw_loop`, since it is necessary to calculate the form of a periodic travelling wave before calculating its spectrum; however this does not increase the computation time significantly, since the majority of the run time is associated with the stability calculation.

To visualise the results, one starts the plotter again by typing

```
plot demo 102
```

(102 is the `p`code value) and then types

```
@pplane
```

at the plotter prompt. If the default plot type (`r`code) is selected, then the resulting plot is illustrated in Figure 2.8; it is very similar to that generated previously by `@pplane` for `p`code value 101, but the colours indicating the existence of periodic travelling waves now also show wave stability. It may be of interest to visualise the eigenvalue spectrum for some of these waves, which can be done using the `@spectrum` plot command. Thus typing

```
@spectrum 1002
```

shows the eigenvalue spectrum for $A = 2.574$ and $c = 1.1$ (illustrated in Figure 2.9a), while

```
@spectrum 1013
```

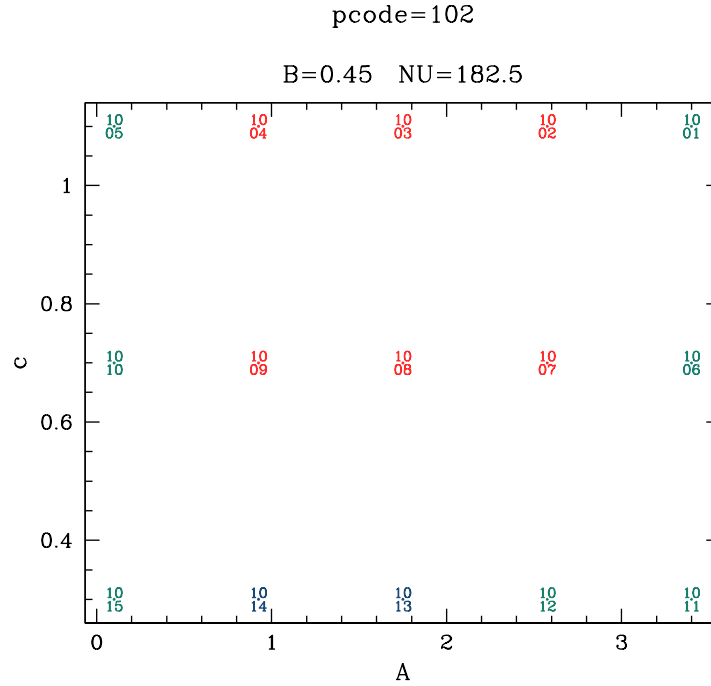


Figure 2.8: Results on the existence and stability of periodic travelling wave solutions for the problem `demo`. A key illustrating the meaning of the various colours is shown in Figure 2.4 on page 18. The run and plot commands used to generate this figure are given in the main text, and are also listed in the Appendix.

shows the spectrum for $A = 1.75$ and $c = 0.3$ (illustrated in Figure 2.9b). (In both cases, the default axes limits are appropriate, and if `gnuplot` is used as the plotter, the default rescalings are appropriate.) In the first case, the wave is stable, and the spectrum remains in the left hand half of the complex plane, while in the second case the wave is unstable, and the spectrum crosses the imaginary axis. As usual, one exits the plotter by typing `@quit` or `@exit`.

The second stage in determining wave stability is to compute the boundary curve(s) between stable and unstable waves. To do this, `WAVETRAIN` requires the coordinates of two points in the A - c parameter plane lying either side of the boundary curve: a periodic travelling wave must exist for the first point, but this is not necessary for the second. As for the `hopf_locus` and `period_contour` commands, the two points must either have the same value of A or the same value of c . Based on the results of the scan through the parameter plane, one can therefore enter the command

```
stability_boundary demo 102 1.75 0.3 1.75 0.7
```

to calculate the stability boundary. This run is significantly more time-consuming than the previous runs (5-10 minutes on a typical desktop computer). During the run, various

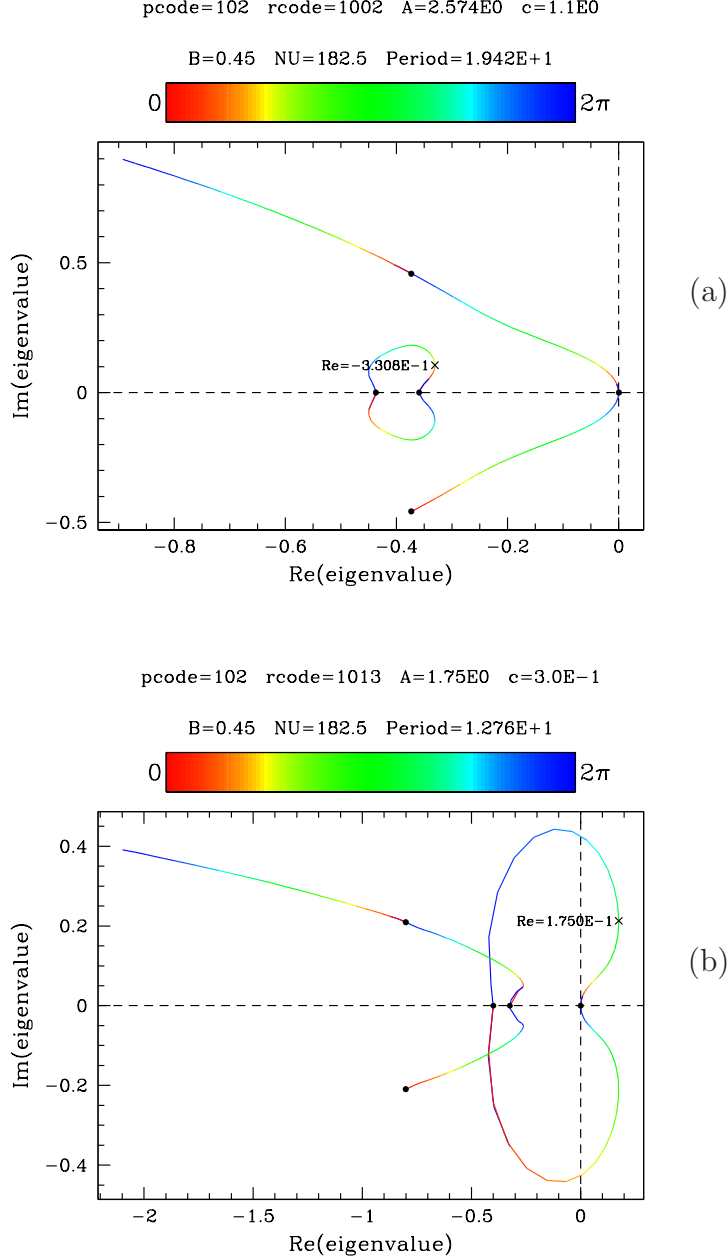


Figure 2.9: The eigenvalue spectrum for two pairs of control parameter and wave speed values, for the problem `demo`. The periodic travelling wave is stable for (a) and unstable for (b). The dots indicate eigenvalues corresponding to periodic eigenfunctions, and the colours along the curve indicate the phase difference in the eigenfunction over one period of the periodic travelling wave. The run and plot commands used to generate these figures are given in the main text, and are also listed in the Appendix. The two parameters B and ν , whose values are given in the titles, appear in the equations but are not being used as the control parameter. Their values are specified in the file `other_parameters.input` (see §2.2.3). Note that if plotting is done using `gnuplot` then ticmarks will be absent from the horizontal axis in these plots; this applies if `spectrumcolour` is set to `rainbow` or `greyscale` or `grayscale`, but not if a single colour is used.

warning messages will appear. The run is given a 3 digit “scode” value (101-999); in this case scode=101 since this is the first stability boundary to be calculated. In general there may be more than one segment of stability boundary in the region of the control parameter–wave speed plane being considered; each can be calculated using different starting coordinates, and they would be allocated different scode values. However, for **demo** there is only one stability boundary curve. Note that the **stability_boundary** command also has optional additional arguments, which are discussed in §3.1.2.

For periodic travelling waves, changes in stability can be of either Eckhaus or Hopf type (see §3.1.14 and Rademacher & Scheel, 2007). The **stability_boundary** command automatically detects which of these occurs. Details of how the **stability_boundary** works are given in §3.1.14, and an example problem with a change in stability of Hopf type is given there; for **demo**, the stability boundary corresponds to a change of Eckhaus type. This is reported in the output information and is available to the plotter, which uses (by default) a different colour for stability boundary curves of the two types. To plot the results one must first generate a parameter plane plot: one restarts the plotter via

```
plot demo 102
```

(102 is the pcode value), and then types

```
@pplane
```

at the plotter prompt, selecting the default plot type. One then superimposes the stability boundary onto this plot via

```
@stability_boundary 101
```

(101 is the scode value); the result is illustrated in Figure 2.10.

The final step in completing the parameter plane plot would be to superimpose the Hopf bifurcation locus and the contours of constant period. These were calculated previously for the same model parameters, but they were computed as part of pcode 101 not pcode 102 and are therefore not currently available to the plotter, which must now be exited via **@quit** or **@exit**. Of course, one could simply repeat the Hopf bifurcation and period contour calculations for pcode 102, but this is not necessary. Instead one can simply type

```
copy_hopf_loci demo 101 102
```

and

```
copy_period_contours demo 101 102
```

(here 101 and 102 are the two pcode values). These commands copy the previously calculated solutions across. A check is made that the two sets of input files are the same, and then all data files copied; note however that the hcode/ccode numbers may be changed.

All calculations for **demo** are now completed. To plot the final result, one starts the plotter again via

```
plot demo 102
```

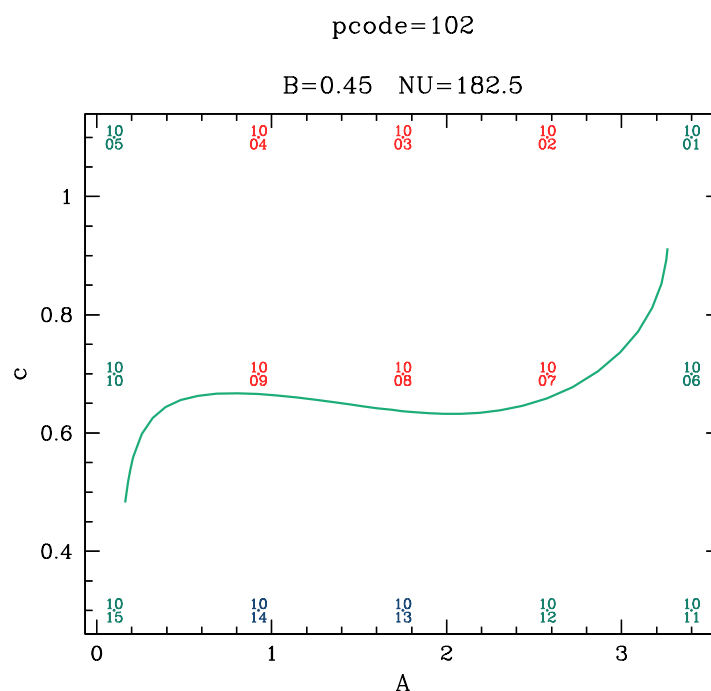


Figure 2.10: The interface between regions of stable and unstable periodic travelling waves, superimposed on results on wave stability, for the problem `demo`. A key illustrating the meaning of the various colours is shown in Figure 2.4 on page 18. The run and plot commands used to generate this figure are given in the main text, and are also listed in the Appendix.

(here 102 is the pcode number) and then enters the relevant series of plot commands:

```
@pplane
```

(to draw the parameter plane, selecting `symbol` as the plot type),

```
@hopf_locus 101
```

(to draw the Hopf bifurcation locus),

```
@period_contour all
```

(to draw the contours of constant period, selecting the default label locations), and

```
@stability_boundary 101
```

(to draw the stability boundary). The resulting plot is illustrated in Figure 2.11. To generate a postscript file of this plot, one types

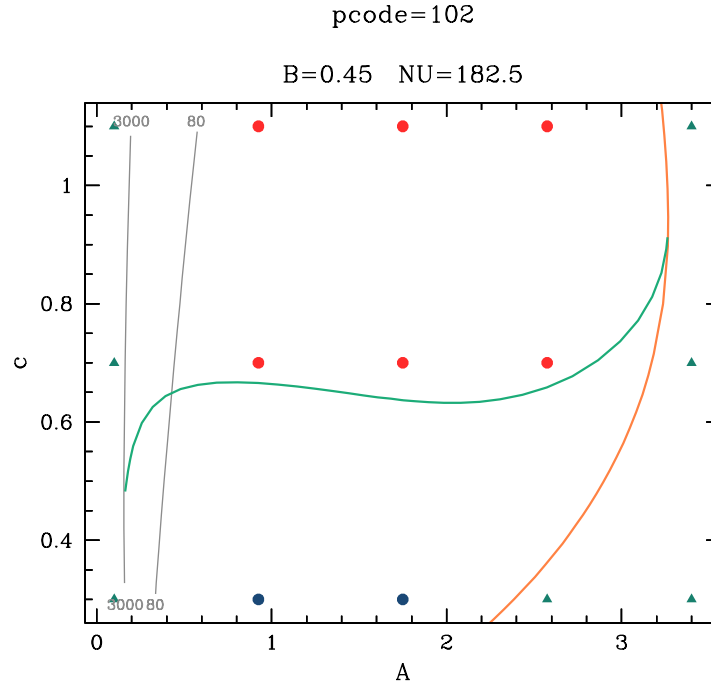


Figure 2.11: A complete plot of the specified part of the control parameter–wave speed plane, for the problem `demo`. A key illustrating the meaning of the various colours and symbols is shown in Figure 2.4 on page 18. The run and plot commands used to generate this figure are given in the main text, and are also listed in the Appendix.

```
@postscript demoplot
```

and then re-enters this series of four plotter commands. However the postscript plot is at this stage only present in a buffer: to complete the file one must either return to screen output via the command `@screen`, or exit the plotter via `@quit` or `@exit`. (Alternatively, one can use the `@postscript` command again to start a new postscript file: this also flushes the output buffer). The plot is then available in the file `demoplot.eps`, in the subdirectory `postscript_files/demo/` of the main `WAVETRAIN` directory. Optionally, one can also give a second argument to the `@postscript` command, which specifies the aspect ratio (horizontal:vertical) of the plot. Thus for example

```
@postscript demoplot 0.3
```

would give a tall, skinny plot. Recall also that a key showing the meaning of the various dots and lines is given in `pplane_key.eps` in the same subdirectory; the key is illustrated in Figure 2.4. Note that each postscript file generated by `WAVETRAIN` contains a single plot: there is no capability to produce multi-page postscript files.

2.2 Details of the WAVETRAIN Input Files

For each problem to be studied using WAVETRAIN, the user must create a subdirectory of `input_files` containing the following files:

`constants.input` This file contains computational constants.

`equations.input` This file specifies the equations to be studied.

`other_parameters.input` (*optional*) This file can be used to specify values for parameters other than the control parameter that occur in the model equations. If there are no such parameters, then the file can either be present but empty (apart from any comments) or can be absent.

`parameter_list.input` This file is only required if the user wishes to run the command `add_points_list`; it specifies a list of control parameter and wave speed values to be studied via this command.

`parameter_range.input` This file specifies the range of control parameter and wave speed values to be studied. It also specifies the number of values of each to be used in a parameter grid for the commands `add_points_loop`, `ptw_loop` or `stability_loop`, and these numbers must be present even if the user does not intend to run any of these commands. If `iwave=0` or `1` (see §3.1.9), then `parameter_range.input` is not needed to run the commands `ptw` and `stability`.

`variables.input` This file specifies the names to be used for the various variables.

For the input directories provided as examples in the installation copy of WAVETRAIN, each of these files contains detailed comments explaining their contents.

In fact, not all computational constants are defined in `constants.input`. Rather, this file contains those constants that it is anticipated users will wish to change according to the problem being studied. Remaining constants are contained in the file `defaults.input` in the `input_files` directory itself. It is not expected that users will need to change the constants in `defaults.input` very often, but the file is available to be edited if desired, and again it contains detailed comments explaining the meaning of each constant. The `input_files` directory also contains a number of plotter input files. The file `plotter_defaults.input` contains the default values for a wide range of plotter settings, while files with the extension “`.style`” specify line styles, colours etc. (see §3.3.1). It is not anticipated that new users will want to change these files, but more experienced users may wish to do so, in order to “fine tune” the plots produced by WAVETRAIN. Both `plotter_defaults.input` and the `.style` files contain detailed comments on each entry.

Full details of the files located in the input subdirectory for a particular problem are given below. Details of the `defaults.input` file are given in §3.2, and details of the `.style` files and the file `plotter_defaults.input` are given in §3.4.

2.2.1 Details of `constants.input`

This file contains those computational constants that are expected to vary according to the problem being studied. Apart from comment lines, the file for the example `demo` is as

follows.

```
1 3 # iclean: 1/2=clean/do not, info: 1/2/3/4=log/error/major/all
1 # iwave, wavefrac (optional)
30 # nmesh1: for accurate ptw calc (for ptw period)
30 # nmesh2: for ptw and eigenvalue calculations (< or = nmesh1)
30 # nmesh3: for spectrum continuation
0.005 0.002 0.05 # ds(>0),dsmin,dsmx for locating Hopf bifns
0.1 0.05 0.5 # ds(>0),dsmin,dsmx for continuation of ptws
0.001 0.001 0.252 # ds(>0),dsmin,dsmx for continuation of spectrum
0.2 0.2 1.0 # ds(>0),dsmin,dsmx for stability boundary calcs
5 0 # nevalues (no. of periodic evals), iposim (0/1=all/only +ve Im)
2 # inttype (1=all ptw components, 2=only pde components)
1 # order: accuracy order of approx of highest spatial derivative
```

Note that here and in all WAVETRAIN input files, # indicates that the remainder of the line is a comment. The meaning of the various constants is as follows.

iclean

WAVETRAIN commands create a variety of temporary files during runs. These are deleted at the end of a run if `iclean=1`, but not if `iclean=2`. Normally `iclean` should be set to 1; the option `iclean=2` may be useful as a diagnostic tool.

info

The level of output information written to the screen is determined by `info`. In all cases, full output information is always written to the file `info.txt` in the relevant output subdirectory. (See §4.4 for a discussion of the output directory structure).

`info=4`: full output information is written to the screen, as well as to `info.txt`.

`info=3`: a subset of information is written to the screen: this provides basic information on the progress of the run.

`info=2`: only error messages are written to the screen, plus an output line indicating the code number allocated to the run.

`info=1`: there is no screen output, and an output line indicating the code number allocated to the run (which is needed for plotting the results) is written to the file `output_files/logfile`.

There two minor exceptions to the above, as follows:

Exception 1: an error may occur that prevents the determination of the value of `info`. For example, there might be a typo in the subdirectory name so that the relevant input subdirectory does not exist. In such a case an error message is written to the screen only.

Exception 2: an error may occur that does not prevent the determination of the value of `info`, but does prevent the creation of the output subdirectory. If `info=1`, an error message is written to `output_files/logfile` as usual in such a case, but if `info=2, 3` or `4` then an error message is written to the screen only.

For interactive runs it is recommended to set `info=3` since `info=4` typically bombards the user with too much information. For background runs it is recommended to set `info=1`.

`iwave`

`iwave>0` causes the program to select the periodic travelling wave at the `iwave`-th occurrence of the specified value of the control parameter along the solution branch. `iwave=0` is similar to `iwave=1` but specifies additionally that, during numerical continuation in the control parameter to search for a Hopf bifurcation in the travelling wave equations, if the specified value of the control parameter is reached before a Hopf bifurcation is detected, then it is assumed that there is no periodic travelling wave for the specified values of the control parameter and wave speed. This has the advantage of eliminating a lot of unnecessary searching for non-existent periodic travelling waves in some cases. It is strongly recommended that when first studying a new problem, `iwave` is set to 1. It is also strongly recommended that `iwave>1` only be used on the basis of a bifurcation diagram, which can be generated via the command `bifurcation_diagram` (see §3.1.6).

`wavefrac`

If `iwave>1`, the program determines the 2nd, 3rd, 4th,...,`iwave`-th occurrence of the specified value of the control parameter along the periodic travelling wave solution branch. At each stage the solution may be a genuine occurrence of an additional wave for that parameter value, or it may be a repeat of the previous wave, caused by the numerical continuation simply re-tracing a previously calculated solution branch. The constant `wavefrac` is used to distinguish between these cases. The new wave solution is deemed to be a repeat of the previous one if the difference divided by the mean is less than `wavefrac` for both the two L2-norms and the two periods. The value of `wavefrac` can be omitted, in which case the default value of 0.001 is used.

`nmesh1`, `nmesh2`, `nmesh3`

The constants `nmesh1`, `nmesh2` and `nmesh3` are the number of mesh intervals used in the discretisation of the solutions, for the three different calculations. These constants are used as the values of the AUTO parameter `ntst`; users familiar with AUTO should note that the AUTO constant `ncol` is set to 4 for all WAVETRAIN computations.

If `nmesh2<nmesh1`, then the periodic travelling wave is calculated twice, first for an accurate value of the periodic travelling wave period, and second for the periodic travelling wave used in the eigenvalue calculations. The lower `nmesh2` value is used simply to reduce computation times. If `nmesh2=nmesh1` then the periodic travelling wave calculation is done only once. Setting `nmesh2>nmesh1` gives an error. It is recommended to set `nmesh1=nmesh2` unless a particularly accurate value of the wave period is needed. Note that if `nmesh2` is set to too small a value, then the first continuation run for the spectrum may fail because the approximations to the periodic eigenfunctions and the corresponding eigenvalues are too poor, due to the coarse discretisation (see §3.1.13).

For both periodic travelling wave calculations, the same value of `nmesh` (`nmesh1` for the first calculation and `nmesh2` for the second) is used for both the location of the Hopf bifurcation point in the periodic travelling wave equations, and the continuation of the periodic travelling wave solution from this Hopf bifurcation point. For calculations of Hopf bifurcation loci, fold loci, and contours of constant period, `nmesh2` is used.

The run time for calculating the eigenvalues of the discretised equations (with periodic boundary conditions) increases rapidly with `nmesh2`, which determines the size of the matrix whose eigenvalues are to be calculated. Therefore large values of `nmesh2` should be used only if necessary (see §3.1.13). If `nmesh2` is too small then two problems arise: (i) spurious eigenvalues, due to the discretisation, with no corresponding eigenvalues in the partial differential equation; (ii) poor approximations to the partial differential equation eigenvalues.

Note that the value of `nmesh2` in `constants.input` is not used by the command `eigenvalue_convergence`. Instead the various values of `nmesh2` specified in the command line are used. However the other parameters set in `constants.input` do apply.

If `nmesh3=nmesh2`, so that the number of collocation points in the spectrum continuation is the same as in the calculation of the periodic travelling wave solution, then the periodic travelling wave solution used in the spectrum continuation is simply inherited directly from the calculated solution. However this is not necessary, and AUTO will interpolate to a new mesh as required. Setting `nmesh3<nmesh2` is usually desirable to reduce run times; setting `nmesh3>nmesh2` is permitted but would rarely be used.

For calculations of stability boundaries, `nmesh3` is used.

`ds`, `dsmin`, `dsmax`

These are the initial, minimum and maximum step sizes used in the various numerical continuation calculations. The signs of `ds` are set appropriately in the relevant programs; they should all be set positive in this file. The appropriate values are highly problem-dependent and some experimentation may be necessary to identify appropriate values. Values that are too small can lead to excessive run times; values that are too large can cause convergence errors or can cause bifurcation points and other critical points to be “jumped over”. The first trio of values is used when locating Hopf bifurcations in the periodic travelling wave equations. The second trio of values is used for continuation of branches of periodic travelling wave solutions, including the calculation of bifurcation diagrams and contours of constant period. They are also used for tracing the loci of Hopf bifurcation points and folds in the control parameter–wave speed plane. However, in the calculation of bifurcation diagrams, Hopf bifurcation loci, fold loci, and contours of constant period, continuations may be done to locate a Hopf bifurcation in the travelling wave equations with the wave speed being the principle continuation parameter rather than the control parameter. (This will occur if the two control parameter values specified in the command arguments are the

same, while those for the wave speed are different). Then the `ds` values used are based on those set in this file (the first trio of values), but with a rescaling. The rescaling is based on the parameter ranges set in `parameter_range.input`: specifically, all three of `ds`, `dsmin` and `dsmax` are multiplied by the scaling factor $(\text{wave_speed_min} + \text{wave_speed_max}) / (\text{control_param_min} + \text{control_param_max})$. However, the values specified in this file are then used for the subsequent continuations, in which the wave speed and control parameter are both continuation parameters.

The third trio of values is used for continuation along the eigenvalue spectrum. When setting these values, it should be remembered that the principle continuation parameter changes by 2π during each continuation. Therefore it is recommended to set `dsmax` no larger than about $2\pi/50 = 0.126$ in this case. The fourth trio of values is used for locating and continuing stability boundaries of both Eckhaus and Hopf type. In many cases these can be set to the same values as for periodic travelling wave continuations. However, since these computations are typically rather time-consuming, larger values should be used if possible. Note that the same step sizes are used to locate the stability boundary whether this is done by varying the control parameter for a fixed wave speed, or vice-versa.

A note for users familiar with AUTO: in the continuations for locating and continuing stability boundaries, derivatives of eigenfunctions are excluded from the pseudo-arclength calculation. Also, in the Eckhaus case, the second derivative of the real part of the eigenvalue with respect to the phase difference across the eigenfunction is excluded, while in the Hopf case, the first derivative of the imaginary part of the eigenvalue with respect to the phase difference is excluded. As a result, it is appropriate to use the same trio of values for both types of stability boundary. However, the pseudo-arclength does include the eigenfunction, the eigenvalue, and the first derivative of the real part of the eigenvalue with respect to the phase difference. This may cause the appropriate step sizes for stability boundary calculations to be different from those for the continuation of periodic travelling wave solution branches.

nevalues

The first stage in the calculation of the spectrum is to find eigenvalues for which the eigenfunction is periodic with the period of the wave (“periodic eigenvalues”). The constant `nevalues` specifies the number of these (chosen in order of real part) to be used as starting points for the calculation of the spectrum itself. The issue is that the spectrum may protrude into the right hand half of the complex plane in between two periodic eigenvalues that both have negative real part. In principle this can happen in between any two adjacent periodic eigenvalues, but it is most common for periodic eigenvalues whose real part is almost maximal. Therefore the recommended value of `nevalues` is about 10.

iposim

If `iposim=1` then only eigenvalues with imaginary part $>$ or $=$ zero are used for output; if `iposim=0` then all eigenvalues are used for output. Often `iposim=1`

is adequate and almost halves the computational time required to calculate the spectrum. However the spectrum can have a form for which the setting `iposim=1` gives incorrect information about stability: a sketch illustrating this is given in the user guide. WAVETRAIN gives an error message in most cases of this kind, informing the user that `iposim=0` is required. However there are unusual situations in which the algorithm used for this error message will fail. Therefore if `iposim=1`, it is good practice to visually inspect spectra.

inttype

When calculating stability boundaries, integral conditions are imposed which ensure that the derivatives of the eigenfunction of the periodic travelling wave with respect to $i\gamma$ are orthogonal to the nullspace of the eigenfunction ordinary differential equations. Here γ is the phase difference in the eigenfunction over one period of the periodic travelling wave. These integral conditions can be formulated either in terms of all of the periodic travelling wave components (given by `inttype=1`), or (in most cases) also in terms of just the periodic travelling wave components that correspond to partial differential equation variables (given by `inttype=2`). The recommended value is `inttype=2` for most problems, which tends to give greater computational reliability (Rademacher *et al.*, 2007). Note that setting `inttype=2` requires that for each of the partial differential equation variables, there is a direct assignment of one of the periodic travelling wave variable to that partial differential equation variable. Thus if one of the partial differential equation variables is u , say, then one of the assignments in `variables.input` must be “<periodic_travelling_wave_variable>=u”.

order

The first step in calculating the eigenvalue spectrum is to discretise the linearised partial differential equations for eigenfunctions that are periodic with the same period as the periodic travelling wave; here the linearisation is about the periodic travelling wave. This gives a matrix eigenvalue problem which is solved to obtain the periodic eigenfunctions and the corresponding eigenvalues. To do this, spatial derivatives must be converted into finite difference approximations. The constant `order` specifies the order of this approximation for the highest spatial derivative. This determines the number of grid points used to represent the derivative. The approximation is calculated using the algorithm of Fornberg (1998). Since the approximation always uses the same number of grid points on either side of the point at which the derivative is being calculated, the order will actually be one higher than that specified if the number of the highest derivative number is odd, and it may be higher anyway if there are particular symmetries. Usually `order = 1` is sufficient, and is the recommended value. However higher values may help the accuracy of the approximations to the periodic eigenfunctions and the corresponding eigenvalues, and hence may help if there are difficulties with convergence in the first AUTO run in the calculation of the spectrum. (This run is a continuation in a dummy parameter to locate a solution of the eigenfunction equations, starting from the discretised periodic eigenfunction and corresponding eigenvalue). Examples of the use of `order > 1` are given in this section (see §2.3.9–2.3.11) and in §3.1.14.

Run times in WAVETRAIN depend strongly on the values of the constants set in the file `constants.input`. Typically run times can be reduced most easily by: (i) decreasing the `nmesh` values, especially `nmesh3`; (ii) increasing `dsmax` for the spectrum continuation; (iii) decreasing `nevalues` (iv) setting `iposim=1` if the form of the spectrum permits this (see §3.1.8).

2.2.2 Details of equations.input

This file specifies the equations to be studied. For the example `demo`, the file is as follows (apart from some comment lines):

```
# Travelling wave equations
U_z=V
V_z=-c*V-W*U*U+B*U
W_z=(W*U*U+W-A)/(NU+c)
#
# Steady states
A<2.001*B
      none
2.001*B<=A
      U=2*B/(A-sqrt(A^2-4*B^2))
      V=0
      W=(A-sqrt(A^2-4*B^2))/2
#
# Wave search method and Hopf bifurcation search range
direct # "direct" / "indirect" : control param / "file"
3.8 # start control param / start speed / soln control param
2.001*B # end control param / end speed / soln speed
#
# Linearised partial differential equations
u_t=
      u_xx & 1
      u    & 2*U*W-B
      w    & U^2
w_t=
      w_x & NU
      u   & -2*W*U
      w   & -1-U**2
#
# The matrix A in the eigenfunction equations
0 & 1 & 0
B-2*U*W & -c & -U^2
2*U*W/(NU+c) & 0 & (1+U**2)/(NU+c)
#
# The matrix B in the eigenfunction equations
```

```

0 & 0 & 0
1 & 0 & 0
0 & 0 & 1/(NU+c)

```

WAVETRAIN converts this file into a series of Fortran77 subroutines. Before describing the various sections of this file, a summary is given of the syntax that is used.

Case sensitivity: Names of variables and parameters are case sensitive.

Derivatives: Derivatives are denoted by `_`. For example, for a partial differential equation variable `u`:

if `x` is the space variable then `u_x` denotes du/dx and `u_xx` denotes d^2u/dx^2 ;
 if `xi` is the space variable then `u_xi` denotes du/dxi and `u_xixi` denotes d^2u/dxi^2 ;
 if `xx` is the space variable then `u_xx` denotes du/dxx and `u_xxxx` denotes d^2u/dxx^2 .
 (It would be rather pathological to choose `xx` as the space variable, but it would not cause difficulties.)

Spaces and blank lines: Spaces and blank lines are allowed and are ignored, but each of the required entries must be on a single line: **line breaks are not allowed**.

Arithmetic:

Powers are denoted by `^` or `**`
 Multiplication is denoted by `*`
 Division is denoted by `/`
 Addition is denoted by `+`
 Subtraction is denoted by `-`
 Brackets must be rounded, `(` and `)`
 Integer division is not allowed. For example `(4/3)` may be evaluated as 1 (integer part) or 1.333 depending on the context. Therefore `4.0/3.0` should be used instead.

Mathematical functions: The following standard mathematical functions are allowed:

`abs`: absolute value.
`acos`: inverse cosine; result between 0 and π .
`asin`: inverse sine; result between $-\pi/2$ and $\pi/2$.
`atan`: inverse tangent; result between $-\pi/2$ and $\pi/2$.
`cos`: cosine, angle in radians.
`cosh`: hyperbolic cosine.
`exp`: exponential function.
`log`: natural logarithm.
`log10`: base 10 logarithm.
`sin`: sine, angle in radians.
`sinh`: hyperbolic sine.
`sqrt`: square root.

step1: step function; result 0, 0, 1 for argument $< 0, = 0, > 0$.
step2: step function; result 0, 0.5, 1 for argument $< 0, = 0, > 0$.
step3: step function; result 0, 1, 1 for argument $< 0, = 0, > 0$.
tan: tangent, angle in radians.
tanh: hyperbolic tangent.

Either upper or lower case can be used for these functions: for example **abs**, **Abs** and **ABS** are equivalent. Note that square roots can be denoted either by $\wedge 0.5$ (or $* 0.5$) or **sqrt** (or **Sqrt** or **SQRT**).

It would be rather confusing to use one of these function names as a variable name, but it is permitted: the conversion to Fortran77 can distinguish between the use of these names as functions, when they must be followed by a left bracket “(”, and as variable names, when they cannot be followed by a left bracket.

Comments: The hash symbol (#) is used to indicate a comment. Any part of a line after a # is ignored.

Having established the syntax, the various sections of the file are now described, in order. Note that it is essential that the sections are given in the correct order.

Travelling wave equations: These are specified using the syntax described above, and using the travelling wave variables and travelling wave coordinate given in the file **variables.input**. It is recommended that the equations are formulated so that the travelling wave variables, the control parameter, and the wave speed all have roughly the same size. Large differences in size can lead to difficulties during numerical continuation: for example, if the principle continuation parameter is much larger than all other quantities, the pseudo-arclength continuation used by AUTO reduces approximately to parameter continuation, which may cause problems if there is a fold in the solution branch being followed.

Steady states: Periodic travelling waves typically develop from a Hopf bifurcation of a steady state solution of the travelling wave equations. In simple cases, this steady state can be specified by a single formula for each travelling wave variable, valid for all values of the control parameter being considered. More generally, the steady state will exist only for some parameter values. Therefore the steady state solution is specified via a series of conditions on parameter values, each of which is followed either by a list of the steady states (using travelling wave variables), or by the single word **none** (or **None** or **NONE**). The parameter conditions must cover all possible values of the control parameter, and must consist of inequalities written using $<$, $>$, \leq and \geq . Multiple inequalities must be written as single inequalities separated by $\&$ (double-sided inequalities are not allowed).

In some cases an exact formula will not be available for the steady state. This is not a problem if a formula for a suitable approximation can be given, since the steady state formulae given in **equations.input** are not used directly: rather, they are used as an initial guess for a numerical calculation of the steady states. However it is often not possible to provide a suitable approximation as a function

of parameters. In this case, the user must enter the single word `file` (or `File` or `FILE`), and must provide an additional input file called `steadystate.input` (in the appropriate input subdirectory). This file should contain a series of control parameter values and associated steady state information, which must be generated by some other computer program. The required format is:

column 1: control parameter value

column 2: 0 or 1, indicating that a steady state does not or does exist

remaining columns, if 1 in column 2: the steady state values, given for the same ordering of the periodic travelling wave variables as in `variables.input`.

The control parameter values must appear in either increasing or decreasing order. This file is read in and then for any specified value of the control parameter, a steady state is estimated by linear interpolation, and then refined via numerical calculation of the steady states. Therefore the control parameter values in `steadystate.input` must span the entire range of values being considered. The interpolation is conservative with respect to steady state existence: if a steady state does not exist for either of the two control parameter values straddling the specified value, then it is assumed not to exist. The problem `demo3`, discussed in §3.1.14, illustrates the provision of the steady state via a user-supplied file. Note that if the “file” wave search method (see below) is used for all control parameter and wave speed values, then the steady states are not used. However a dummy specification is still required; this can be “file”, and then no `steadystate.input` file is needed.

A common situation is that the steady state exists only on one side of a parameter threshold, with a saddle-node bifurcation at the threshold. In this case, there will be two steady states on one side of the threshold, only one of which is of interest (in any particular run). If the exact saddle-node point is specified in the parameter inequalities, then it is quite possible that, starting from this saddle-node point, WAVETRAIN will numerically continue the steady state along the wrong steady state branch. To avoid this, the user is strongly recommended to define the steady state as not existing for parameter values very close to the threshold.

Wave search method and Hopf bifurcation search range: Typically, periodic travelling waves develop from a Hopf bifurcation of the steady state specified in the previous part of `equations.input`. The next part of the file specifies the method used to search for periodic travelling waves, and the associated range of parameters used to search for the initial Hopf bifurcation point. Three methods are available. Typically, periodic travelling waves develop from a Hopf bifurcation of the steady state specified in the previous part of this input file, and two of these methods involve first locating a Hopf bifurcation in the steady state. These methods are termed “direct” and “indirect” because after detecting the Hopf bifurcation, WAVETRAIN locates the periodic travelling wave in either one or two steps, respectively. The third available method is termed “file”, and does not involve locating a Hopf bifurcation point. Rather, the user is required to provide a periodic travelling wave solution for one pair of

control parameter and wave speed values (e.g. from simulations of the partial differential equations). This solution is then used as a starting point for calculating wave solutions for other control parameter and wave speed values.

Direct method: In this case the user specifies a search range of control parameter values. WAVETRAIN will then search this range for a Hopf bifurcation in the travelling wave equations, with the wave speed fixed at the value for which a periodic travelling wave is sought. Having found a Hopf bifurcation, WAVETRAIN then tracks the branch of periodic travelling waves as the control parameter varies, again with fixed wave speed, until the required control parameter value is reached.

Indirect method: In this case the user specifies a search range of wave speed values, and a control parameter value for the search. WAVETRAIN will then search for a Hopf bifurcation on this basis. Having found a Hopf bifurcation, it will first track the branch of periodic travelling waves as the wave speed varies, with the control parameter fixed at the value specified for the Hopf bifurcation search, until the required wave speed is reached. WAVETRAIN then fixes the wave speed at this value and tracks the periodic travelling waves as the control parameter is varied, until the required value is reached.

File method: In this the user specifies a pair of control parameter and wave speed values, and provides an additional input file `ptwsoln.input` that contains the solution for these parameter values. The file must be placed in the relevant input subdirectory. Using this solution as a starting point, WAVETRAIN will first track the branch of periodic travelling waves as the wave speed varies, with the control parameter fixed at the value specified for the Hopf bifurcation search, until the required wave speed is reached. WAVETRAIN then fixes the wave speed at this value and tracks the periodic travelling waves as the control parameter is varied, until the required value is reached. The required format for the file `ptwsoln.input` is:

column 1: travelling wave (or space) coordinate

remaining columns: the travelling wave variables, in the same order as listed in `variables.input`.

There are no particular constraints on the discretisation used for the solution in the file. It must be given over one period of the periodic travelling wave, and the travelling wave variable values given in the last line must be exactly the same as those given in the first line. The corresponding space/travelling wave coordinate will then be one period greater than that in the first line: usually this coordinate will be an approximation, found by linear interpolation.

In simple cases a single method will be specified for all control parameter and wave speed values, but the choice and details of the method can be parameter dependent, specified using inequalities with the same syntax as described above for steady states. Thus the parameter conditions (which must cover all values of the control parameter and wave speed being studied) must consist of inequalities written using `<`, `>`, `<=`

and \geq . Multiple inequalities must be written as single inequalities separated by $\&$ (double-sided inequalities are not allowed). The following points should be noted in connection with the specification of the wave search method and Hopf bifurcation search range:

1. (direct and indirect cases) The start value(s) of control parameter/wave speed can be either less than or greater than the end value(s).
2. (direct and indirect cases) The steady state must exist throughout the search range(s).
3. (direct and indirect cases) The start and end values can depend on the control parameter and/or the wave speed.
4. (direct and indirect cases) Choosing range limits that are closer together (and thus closer to the Hopf bifurcation point) will speed up execution.
5. The setting `iwave=0` (§3.1.9) is only permissible if the search method(s) is/are direct. Then the end value of the search range (for the control parameter) is not used, although a (dummy) value must still be given; it is replaced by the value of the control parameter at which a periodic travelling wave is sought. (This is the only difference between `iwave=0` and `iwave=1`).
6. The start and end values of the search range (direct and indirect cases) and the specific control parameter and wave speed value (file case) can lie outside the limits specified in `parameter_range.input`.
7. (direct and indirect cases) The Hopf bifurcation search range that is specified in `equations.input` is not used by either the `hopf_locus` command or the `bifurcation_diagram` command. In the former case, the search range is specified in the command arguments, and for the `bifurcation_diagram` command the limits specified in the file `parameter_range.input` are used to search for Hopf bifurcations.
8. (indirect case) Note the semi-colon after the word `indirect`, which is essential.

Linearised partial differential equations: The first step in the calculation of the eigenvalue spectrum is to calculate the eigenvalues corresponding to eigenfunctions that are periodic with the same period as the travelling wave solution. To do this, one first linearises the partial differential equations around the periodic travelling wave solution, and then discretises this in space to give a matrix eigenvalue problem. The discretisation is done automatically by WAVETRAIN, but the linearised partial differential equations must be specified. If the time variable is \mathbf{t} , the partial differential equation variables are $\mathbf{u}, \mathbf{v}, \mathbf{w}, \dots$ and the periodic travelling wave variables are $\mathbf{U}, \mathbf{V}, \mathbf{W}, \dots$ (say), then each linearised equation will have either $\mathbf{u}_\mathbf{t}, \mathbf{v}_\mathbf{t}, \mathbf{w}_\mathbf{t}, \dots$ or 0 on the left hand side of the equation, and a series of terms on the right hand side, each of which contains exactly one of $\mathbf{u}, \mathbf{v}, \mathbf{w}, \dots$ or a spatial derivative of one of these partial differential equation variables. The required format for inputting these equations is $\mathbf{u}_\mathbf{t} =$ (or $\mathbf{v}_\mathbf{t} =$ or $\mathbf{w}_\mathbf{t} =$ or \dots) or $0 =$ on a separate line, and then on subsequent lines the partial differential equation variable or its derivative, followed by

&, and then the coefficient of this partial differential equation variable/derivative. Each pair of partial differential equation variable/derivative and coefficient must be on a separate line. The coefficients must involve just parameters and travelling wave variables, not the derivatives of travelling wave variables. (If such derivatives arise when the partial differential equations are linearised, these can always be expressed in terms of other travelling wave variables). Note that the equations should be given in terms of the space variable, rather than the travelling wave coordinate. They are converted to a moving reference frame before being discretised.

Eigenfunction equations: Investigation of the stability of periodic travelling waves requires solution of the eigenfunction equations that govern behaviour close to the wave solution. These can be specified by two $\text{NPTW} \times \text{NPTW}$ matrices (where NPTW is the number of travelling wave variables). To derive these matrices it will be necessary to do a few lines of mathematics on a piece of paper, but this should not be difficult or time-consuming. The procedure is as follows:

Step 1: Linearise the partial differential equations about the periodic travelling wave solution, which has been done for the previous part of `equations.input`. The result of this calculation is a linear system of partial differential equations given above. To fix notation, denote the linearised partial differential equation variables by the NPDE-dimensional vector $\underline{u}_{lin}(\mathbf{x}, \mathbf{t})$. Here \mathbf{x} and \mathbf{t} are the space and time variables, and NPDE is the number of partial differential equation variables.

Step 2: Substitute the solution form $\underline{u}_{lin}(\mathbf{x}, \mathbf{t}) = \underline{U}_{lin}(\mathbf{z}) \exp(\lambda \mathbf{t})$ where $\mathbf{z} = \mathbf{x} - \mathbf{c}\mathbf{t}$ is the travelling wave variable with \mathbf{c} denoting the wave speed. This gives a linear system of ordinary differential equations for \underline{U}_{lin} , whose coefficients depend on the travelling wave solution.

Step 3: Typically this linear system of ordinary differential equations will involve second and/or higher order derivatives of \underline{U}_{lin} . In the file `variables.input`, the user is required to provide formulae for converting the partial differential equation variables into travelling wave variables. To be specific, denote these formulae by $\underline{u}_{tw} = F(\underline{u}_{pde}, \partial \underline{u}_{pde} / \partial \mathbf{x}, \partial^2 \underline{u}_{pde} / \partial \mathbf{x}^2, \dots)$. Then the corresponding formulae $\underline{U}_{lin,tw} = F(\underline{U}_{lin}, \partial \underline{U}_{lin} / \partial \mathbf{z}, \partial^2 \underline{U}_{lin} / \partial \mathbf{z}^2, \dots)$ should be used to convert the linear ordinary differential equations into a first order system of ordinary differential equations for $\underline{U}_{lin,tw}$. **Important note:** it is essential that the components of the vector $\underline{U}_{lin,tw}$ are kept in the order corresponding to the order in which the travelling wave variables are listed in the file `variables.input`.

Step 4: The resulting first order linear system of differential equations will always have the form $\partial \underline{U}_{lin,tw} / \partial \mathbf{z} = A \underline{U}_{lin,tw} + \lambda B \underline{U}_{lin,tw}$ where A and B are $\text{NPTW} \times \text{NPTW}$ matrices.

The final two sections of the file specify the matrices A and B , with columns separated by ampersands (&).

2.2.3 Details of `other_parameters.input`

This file can be used to specify values for parameters other than the control parameter that occur in the model equations. Apart from comment lines, the file for the example `demo` is as follows.

```
B=0.45      # <name> = <value>
NU=182.5
```

Spaces and comments (indicated by `#`) can be included in this file if desired: they will simply be ignored. Note that if the partial differential equations being studied do not contain any parameters other than the control parameter then this file can either be absent, or can exist but be empty, or can exist but have only comment lines.

2.2.4 Details of `parameter_list.input`

This file is only required if the user wishes to run the `add_points_list` command; it specifies a list of control parameter and wave speed values to be studied via this command. Apart from comment lines, the file for the example `demo` is as follows.

```
2.0 0.61
2.0 0.65
0.2 0.7
0.4 0.7
```

The first column are control parameter values and the second column are wave speed values. For each pair of values, the command `add_points_list` calculates the periodic travelling wave; stability is also calculated if the `pcode` value in the argument list was generated using the `stability_loop` command. The file can, if desired, contain points outside the range specified in `parameter_range.input` at the time of the initial run of `ptw_loop` or `stability_loop`. In this case the axes limits in subsequent control parameter–wave speed plots will be expanded to include the additional points.

2.2.5 Details of `parameter_range.input`

This file specifies the range of control parameter and wave speed values to be studied. It also specifies the number of values of each to be used in a parameter grid for the commands `add_points_loop`, `ptw_loop` or `stability_loop`, and the numbers must be present even if the user does not intend to run any of these commands. Apart from comment lines, the file for the example `demo` is as follows.

```
0.1 3.4 5 # min,max,ngid for control parameter
0.3 1.1 3 # min,max,ngid for wave speed
```

The specified minimum and maximum values are inclusive. The file is also used when calculating bifurcation diagrams, providing diagram limits for whichever of the control parameter or wave speed is the bifurcation parameter. If `iwave=0` or `1`, then the commands `ptw` and `stability` do not require limits on the control parameter and wave speed values, and the file `parameter_range.input` is not required. However even these two commands require `parameter_range.input` for `iwave≥2`, since it provides constraints on the ranges within which WAVETRAIN tracks the periodic travelling wave solution branch, looking for

folds. When running the command `add_points_loop`, the new control parameter and wave speed ranges can, if desired, extend beyond those specified in `parameter_range.input` at the time of the initial run of `ptw_loop` or `stability_loop`. In this case the axes limits in subsequent control parameter–wave speed plots will be expanded to include the additional points.

2.2.6 Details of `variables.input`

This file specifies the names to be used for the various variables. It contains comments that explain what the various lines denote. For the example `demo`, the file is as follows (apart from some explanatory comments):

```
# variables.input
#####
A # NAME FOR THE CONTROL PARAMETER
c # NAME FOR THE WAVE SPEED
x # NAME FOR THE SPACE COORDINATE
t # NAME FOR THE TIME COORDINATE
z # NAME FOR THE TRAVELLING WAVE COORDINATE
#####
# PDE VARIABLE NAMES, ONE PER LINE
u
w
#####
# TW VARIABLE NAME = FUNCTION OF PDE VARIABLES, ONE PER LINE
U = u
V = u_x
W = w
```

It is essential that the various lines in this file are given in the correct order. Spaces, blank lines and comments (indicated by `#`) are ignored. Variables can have any names, with any number of characters, except that the words “file” and “variable” are not permitted as either partial differential equation or travelling wave variables; a test for this is made at the start of commands.

2.3 A Worked Example

Together, §2.1 and §2.2 describe the basic operation of WAVETRAIN, and the details of most of the input files. A number of details of the commands have not been discussed, and a few WAVETRAIN commands have not been mentioned at all; these are considered in §3.1. Before this, I present a full worked example, illustrating how WAVETRAIN can be used to study periodic travelling wave solutions in practice. The example is based on the work of Smith & Sherratt (2007), and concerns the following model for a predator-prey interaction:

$$\frac{\partial u}{\partial t} = e^{\delta} \frac{\partial^2 u}{\partial x^2} + u(1 - u) - \frac{uv}{u + k} \quad (2.1a)$$

$$\frac{\partial v}{\partial t} = e^{-\delta} \frac{\partial^2 v}{\partial x^2} + \frac{su v}{u + k} - \mu v. \quad (2.1b)$$

Here u and v denote prey and predator densities respectively, x and t are the space and time coordinates respectively, and k , s , μ and δ are positive parameters. Based on the numerical simulations of (2.1) (see Smith & Sherratt, 2007), the objective is to investigate the existence and stability of periodic travelling waves for the parameter δ in the range $-2 < \delta < 2$, with wave speeds between 0.1 and 4, and with the other three parameters fixed at $k = 0.2$, $s = 0.15$ and $\mu = 0.05$. Note that periodic travelling waves do exist for wave speeds between 0.1 and 0, and these can be investigated using WAVETRAIN. However the computations become significantly more difficult as the speed approaches zero, requiring very small continuation step sizes, which leads to rather long run times.

For ease of presentation, the study is broken up into a number of stages, some involving the creation or replacement of input files, and others involving runs of WAVETRAIN commands. Deliberately, the installation files do not include an input subdirectory corresponding to this example. This is because the example is intended to illustrate all of the stages involved in a WAVETRAIN study, which includes the creation of an input subdirectory. Users may wish to perform each stage of this example manually, to help familiarise themselves with the use of WAVETRAIN. However, as an alternative, the command `set_worked_example_inputs` is provided. This command takes as an argument the number of a stage of the study involving input file creation/replacement. It first empties the `workedx` input subdirectory, and then performs all of the steps involved in each of the input stages, in order, up to and including the specified stage. Thus for example the command

```
set_worked_example_inputs 5
```

will first empty the `workedx` input subdirectory, if it exists, and will then perform the various steps in Stage 1, then Stage 3, and finally Stage 5.

2.3.1 Stage 1 (Input): Create the Initial Input Files

The command “set_worked_example_inputs 1” performs all of the steps described in this stage of the study.

2.3.1.1 Create the Input Subdirectory

The very first step in any WAVETRAIN study is to choose a problem name and create the input directory. There is no constraint on the length of the name, although names with more than 10 characters are abbreviated in screen output. In this case, the name “workedx” seems appropriate, and the input directory is created by the command

```
mkdir input_files/workedx
```

which must be entered from the main WAVETRAIN directory. At installation, a dummy set of input files is provided in the input directory `template`, and this should be copied into the new input directory via the command

```
cp input_files/template/* input_files/workedx/.
```

(for the benefit of users unfamiliar with unix, it may be helpful to explain that the final dot in this command indicates that the files should be given the same names in the new directory as in `template`).

2.3.1.2 Create the File `equations.input`

To create an initial version of `equations.input`, the various parts of the file should be completed in the order described in §2.2.2, beginning with the travelling wave equations. Substituting $u(x, t) = U(z)$ and $v(x, t) = V(z)$ with $z = x - ct$ into (2.1) gives

$$\begin{aligned} e^\delta U'' + cU' + U(1 - U) - UV/(U + k) &= 0 \\ e^{-\delta} V'' + cV' + sUV/(U + k) - \mu V &= 0 \end{aligned}$$

where prime denotes d/dz . This must be rewritten as a system of first order equations:

$$U' = P \tag{2.2a}$$

$$P' = e^{-\delta} [-cP - U(1 - U) + UV/(U + k)] \tag{2.2b}$$

$$V' = Q \tag{2.2c}$$

$$Q' = e^\delta [-cQ - sUV/(U + k) + \mu V] . \tag{2.2d}$$

Setting the left hand sides of (2.2) to zero provides a system of algebraic equations that can easily be solved for the unique steady state with U and V both non-zero:

$$U_s = \mu k / (s - \mu) \quad P_s = 0 \quad V_s = (1 - U_s)(U_s + k) \quad Q_s = 0 . \tag{2.3}$$

After calculating the travelling wave equations and steady state, the next part of `equations.input` concerns the wave search method. At this stage no information is available on which this can be based, and therefore the dummy entries provided in the version of the file in the `template` directory should be retained. Note that dummy entries are essential: a blank entry is not permitted.

Moving on to the linearised partial differential equations, these are given by linearising (2.1) about the travelling wave solution $u(x, t) = U(z)$, $v(x, t) = V(z)$, giving

$$\frac{\partial u_{lin}}{\partial t} = e^\delta \frac{\partial^2 u_{lin}}{\partial x^2} + u_{lin} \left[1 - 2U - \frac{kV}{(U + k)^2} \right] + v_{lin} \left[\frac{-U}{U + k} \right] \tag{2.4a}$$

$$\frac{\partial v_{lin}}{\partial t} = e^{-\delta} \frac{\partial^2 v_{lin}}{\partial x^2} + u_{lin} \left[\frac{skV}{(U + k)^2} \right] + v_{lin} \left[\frac{sU}{U + k} - \mu \right] \tag{2.4b}$$

where $u_{lin}(x, t) = u(x, t) - U(z)$ and $v_{lin}(x, t) = v(x, t) - V(z)$, and nonlinear terms have been neglected.

Finally, it is necessary to calculate the matrices A and B in the eigenfunction equations, via the method described in §2.2.2. Substituting $u_{lin}(x, t) = U_{lin}(z)e^{\lambda t}$ into (2.4) gives

$$\begin{aligned} \lambda U_{lin} &= e^\delta \frac{d^2 U_{lin}}{dz^2} + c \frac{dU_{lin}}{dz} + U_{lin} \left[1 - 2U - \frac{kV}{(U + k)^2} \right] + V_{lin} \left[\frac{-U}{U + k} \right] \\ \lambda V_{lin} &= e^{-\delta} \frac{d^2 V_{lin}}{dz^2} + c \frac{dV_{lin}}{dz} + U_{lin} \left[\frac{skV}{(U + k)^2} \right] + V_{lin} \left[\frac{sU}{U + k} - \mu \right] . \end{aligned}$$

Substituting

$$U_{lin} = U_{lin,tw} \quad dU_{lin}/dz = P_{lin,tw} \quad V_{lin} = V_{lin,tw} \quad dV_{lin}/dz = Q_{lin,tw} \quad (2.5)$$

gives a system of the required form

$$\frac{d}{dz} \begin{bmatrix} U_{lin,tw} \\ P_{lin,tw} \\ V_{lin,tw} \\ Q_{lin,tw} \end{bmatrix} = (A + \lambda B) \begin{bmatrix} U_{lin,tw} \\ P_{lin,tw} \\ V_{lin,tw} \\ Q_{lin,tw} \end{bmatrix}$$

where A and B are the 4×4 matrices

$$A = \begin{bmatrix} 0 & 1 & 0 & 0 \\ e^{-\delta} \left(2U + \frac{kV}{(U+k)^2} - 1 \right) & -e^{-\delta}c & e^{-\delta} \frac{U}{U+k} & 0 \\ 0 & 0 & 0 & 1 \\ -e^{\delta} \frac{skV}{(U+k)^2} & 0 & e^{\delta} \left(\mu - \frac{sU}{U+k} \right) & -e^{\delta}c \end{bmatrix} \quad (2.6a)$$

$$B = \begin{bmatrix} 0 & 0 & 0 & 0 \\ e^{-\delta} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & e^{\delta} & 0 \end{bmatrix}. \quad (2.6b)$$

Equations (2.2), (2.3), (2.4) and (2.6) must be entered in the `equations.input` file, using the format described in §2.2.2, and also in the comments at the end of the file. Apart from explanatory comments at the end, the resulting file is as follows:

```
# equations.input
#####
# Travelling wave equations
U_z=P
P_z=exp(-delta)*(-c*P-U*(1-U)+U*V/(U+k))
V_z=Q
Q_z=exp(delta)*(-c*Q-s*U*V/(U+k)+mu*V)
#####
# Steady states
U=mu*k/(s-mu)
P=0
V=(1-(mu*k/(s-mu)))*((mu*k/(s-mu))+k)
Q=0
#####
# Wave search method and Hopf bifurcation search range
direct # "direct" / "indirect" : control param / "file"
0.0 # start control param / start speed / soln control param
0.0 # end control param / end speed / soln speed
#####
# Linearised partial differential equations
u_t=
```

```

u_xx & exp(delta)
u & 1-2*U-k*V/((U+k)**2)
v & -U/(U+k)
v_t=
v_xx & exp(-delta)
u & s*k*V/((U+k)**2)
v & s*U/(U+k)-mu
#####
# The matrix A in the eigenfunction equations
0 & 1 & 0 & 0
exp(-delta)*(2*U+k*V/((U+k)**2)-1) & -exp(-delta)*c & exp(-delta)*U/(U+k) & 0
0 & 0 & 0 & 1
-exp(delta)*s*k*V/((U+k)**2) & 0 & exp(delta)*(mu-s*U/(U+k)) & -exp(delta)*c
#####
# The matrix B in the eigenfunction equations
0 & 0 & 0 & 0
exp(-delta) & 0 & 0 & 0
0 & 0 & 0 & 0
0 & 0 & exp(delta) & 0

```

Note that although two of the rows of the matrix *A* cause rather long lines, they must be entered on a single line: line breaks are not permitted in `equations.input`.

2.3.1.3 Create the File `variables.input`

The various parts of `variables.input` must now be completed, giving the file as follows (apart from the explanatory comments at the end of the file):

```

# variables.input
#####
delta  # NAME FOR THE CONTROL PARAMETER
c      # NAME FOR THE WAVE SPEED
x      # NAME FOR THE SPACE COORDINATE
t      # NAME FOR THE TIME COORDINATE
z      # NAME FOR THE TRAVELLING WAVE COORDINATE
#####
# PDE VARIABLE NAMES, ONE PER LINE
u
v
#####
# TW VARIABLE NAME = FUNCTION OF PDE VARIABLES, ONE PER LINE
U = u
P = u_x
V = v
Q = v_x

```

Note that it is essential that the order in which the travelling wave variables are listed corresponds to the order used for the matrices *A* and *B* in `equations.input`.

2.3.1.4 Create the File `other_parameters.input`

This file is simply a listing of the parameters other than the control parameter, and their required values. Apart from explanatory comments at the end, the file is:

```

k=0.2 # <name> = <value>

```

```
s=0.15
mu=0.05
```

2.3.1.5 Create the File `parameter_range.input`

This file specifies the range of control parameter (δ) and wave speed (c) values to be considered. It also specifies the numbers of grid points used for scans of the control parameter – wave speed plane. This latter information is not needed at this stage, but it is convenient to enter appropriate values, to save changing the file later: an 8×8 grid is suitable, and would be an appropriate initial choice for many problems. Apart from explanatory comments at the end, the file is therefore:

```
-2 2 8 # min,max,ngrid for control parameter
0.1 3 8 # min,max,ngrid for wave speed
```

2.3.1.6 Create the File `constants.input`

In the directory `template`, the various entries in `constants.input` are given values that are good starting points for many problems. Note that the entries are different from those used for the problem `demo`, which were effective for demonstration purposes but would not be suitable for many real problems. Therefore at this stage, the version of `constants.input` that has been copied from the `template` directory can be used without changes.

2.3.2 Stage 2 (Run): Formulate a Wave Search Method

The input files generated in Stage 1 lack one key ingredient: a wave search method was not specified in `equations.input`. The key tool for formulating a method is the `hopf_locus` command, which does not use the wave search method part of `equations.input`. However, this command requires a pcode value as one of its arguments. In §2.1, `hopf_locus` was run after a run of `ptw_loop` or `stability_loop` so that a pcode value was available, but that is not the case here. The WAVETRAIN command `new_pcode` is provided for this purpose. This command will be described in §3.1.5, but it is very simple. It takes only one argument, the input directory name, sets up a blank control parameter – wave speed plane, and allocates a pcode value. Therefore, to begin investigation of the `workedex` problem, one types

```
new_pcode workedex
```

and WAVETRAIN reports that the allocated pcode value is 101.

The `hopf_locus` command can now be run. In the absence of any information about control parameter and wave speed values for which the travelling wave equations might have a Hopf bifurcation, it is necessary to experiment blindly. Typing

```
hopf_locus workedex 101 -1 0 -1 6
```

causes WAVETRAIN to report that it did not detect a Hopf bifurcation in the range $0 < c < 6$ for $\delta = -1$. Here the range of c being investigated by the command is wider than the range specified in `parameter_range.input`, which is permitted. Note also that the

run is allocated an hcode value even though it is unsuccessful. It is of course possible that a typo might be present in `equations.input`. This would first be manifested via the above command, being the first time that the file is used; often the result would be a large number of system errors. It is important to note that after fixing the typo, the `new_pcode` command would have to be rerun. This is because for any WAVETRAIN command in which a pcode value ($\neq 100$) is specified, the `equations.input` file is inherited from the time at which the pcode was initially allocated.

As a second attempt to find a Hopf bifurcation locus, type

```
hopf_locus workedex 101 1 0 1 6
```

which does successfully locate a Hopf bifurcation and trace its locus. To plot this, start the plotter by typing

```
plot workedex 101
```

and plot the locus by entering first

```
@pplane
```

and then

```
@hopf_locus 102
```

at the plotter prompt. (Here 102 is the hcode value of the successful run). The resulting plot is shown in Figure 2.12. One can now exit the plotter by typing `@exit` or `@quit`.

2.3.3 Stage 3 (Input): Add a Preliminary Wave Search Method

The command “set_worked_example_inputs 3” performs the steps described in Stage 1 of the study, and then performs the steps described in this stage.

From the plot of the Hopf bifurcation locus in Figure 2.12 one cannot tell whether periodic travelling waves exist for parameters above or below the locus. Indeed it is possible that both of these possibilities apply, because of folds in the periodic travelling wave solution branches. In fact, numerical simulations of the partial differential equations (Smith & Sherratt, 2007) suggest that waves exist only above the locus, but this can be determined using WAVETRAIN, without *a priori* knowledge. To do this, one must formulate a preliminary wave search method that will apply for $0.1 < c < 1.0$; this will be amended to cover larger values of c in Stage 5. There is a Hopf bifurcation in the travelling wave equations for every value of c between 0.1 and 1, with the corresponding value of δ lying between -1 and 2 . Therefore the file `equations.input` can be edited, replacing the dummy wave search method with

```
# Wave search method and Hopf bifurcation search range
direct # "direct" / "indirect" : control param / "file"
-1.2 # start control param / start speed / soln control param
2.2 # end control param / end speed / soln speed
```

Note that the search range has been set slightly wider than strictly necessary; this is good practice to avoid problems due to large jumps in the first few continuation steps.

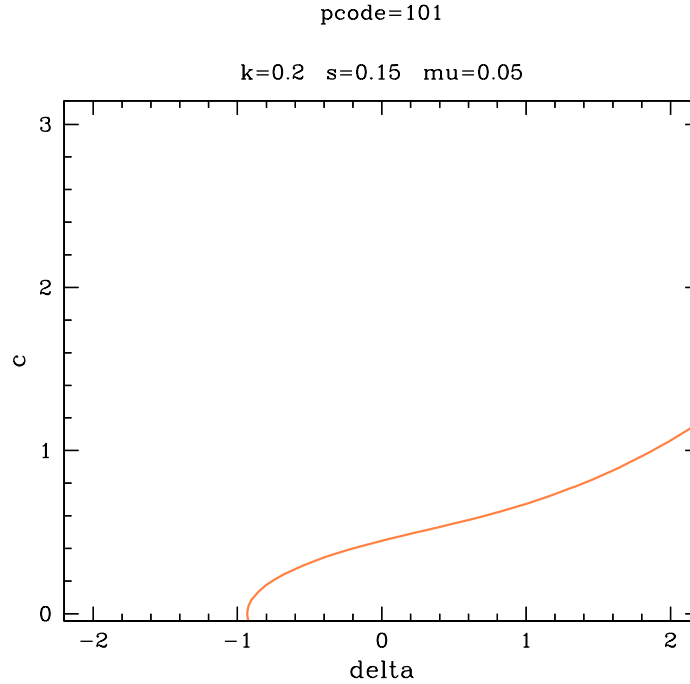


Figure 2.12: A plot of the locus of Hopf bifurcation points in the control parameter–wave speed plane, for the worked example problem `workedex`. The run and plot commands used to generate this figure are given in the main text, and are also listed in the Appendix.

2.3.4 Stage 4 (Run): Look for Waves near the Hopf Bifurcation Locus

With the preliminary wave search method in place, one can look for periodic travelling waves on either side of the Hopf bifurcation locus. For example, the run

```
ptw workedex 0 0.7
```

successfully detects a periodic travelling wave, while

```
ptw workedex 1.0 0.3
```

fails to find a wave solution. Based on this and other similar runs (which must have c between 0.1 and 1) one can draw the (tentative) conclusion that close to the Hopf bifurcation locus, waves exist above the locus and not below.

2.3.5 Stage 5 (Input): Amend the Wave Search Method

The command “`set_worked_example_inputs 5`” performs the steps described in the previous input stages of the study, and then performs the steps described in this stage.

Based on the results of Stage 4, it is now necessary to amend the wave search method to cover all points in the parameter plane under study that lie above the Hopf bifurcation locus. One possible approach to this would be to calculate the Hopf bifurcation locus over a large range of δ values. Depending on the shape of the locus, it might then be possible to formulate a single direct search method: note that parameter ranges used in the wave search method can be wider than those specified in `parameter_range.input`. However, it is simpler to formulate a mixed search method based only on the results in Figure 2.12. An appropriate choice is:

```
# Wave search method and Hopf bifurcation search range
c<=1
  direct # "direct" / "indirect" : control param / "file"
  -1.2 # start control param / start speed / soln control param
  2.2 # end control param / end speed / soln speed
c>1
  indirect : 0.0 # "direct" / "indirect" : control param / "file"
  0.0 # start control param / start speed / soln control param
  0.9 # end control param / end speed / soln speed
```

Note that in the indirect case there is nothing special about the choice of 0.0 as the fixed value of δ for the Hopf bifurcation search, nor is there anything special about the range $[0.0, 0.9]$ for c : all that is required is that there is a Hopf bifurcation for c in this range when $\delta = 0$.

2.3.6 Stage 6 (Run): Test Runs of the `ptw` Command

Having set up the wave search method, it is necessary to test the `ptw` command at a number of control parameter and wave speed values. The objective is to check whether the computational constants set in `constants.input` (and potentially also in `defaults.input`) are suitable for this problem. As examples, the commands

```
ptw workedex 1.5 2.5
ptw workedex 2.0 1.2
ptw workedex 0.5 4.0
ptw workedex -1.0 0.7
ptw workedex -2.0 0.2
```

all successfully find periodic travelling waves. In the third case a warning is reported about a small number of continuation steps. This is because $\delta = 0.5$ is rather close to the value fixed for the Hopf bifurcation search ($\delta = 0$), and the warning is not a cause for concern. However, the command

```
ptw workedex -1.5 0.1
```

unexpectedly fails to find a travelling wave, due to a convergence problem during the periodic travelling wave continuation. As explained in the screen output, this type of error commonly occurs because the wave solution branch is approaching a homoclinic solution. (An example of this is provided by the problem `demo` discussed in §2.1). However the warning message that only two continuation steps were taken strongly suggests that a

real convergence error has occurred. Users with previous experience with AUTO may find it valuable to look at the `info.txt` file associated with the run in a situation such as this; `info.txt` contains the full output from AUTO (see §2.5 for details).

2.3.7 Stage 7 (Input): Alter the File `constants.input`

The command “`set_worked_example_inputs 7`” performs the steps described in the previous input stages of the study, and then performs the steps described in this stage.

There are two likely remedies for a convergence error such as that encountered in Stage 6: either an increase in the values of `nmesh1` and `nmesh2`, or a decrease in the values of `ds` and `dsmin`. Experimentation shows that increasing `nmesh1` and `nmesh2` (keeping them equal for convenience) does not eliminate the error, but reducing `ds` and `dsmin` does. Therefore line 7 of `constants.input` should be altered, with appropriate new settings being

```
0.01 0.01 0.5 # ds(>0),dsmin,dsmx for continuation of ptws
```

2.3.8 Stage 8 (Run): Further Test Runs of `ptw`, and a Run of `ptw_loop`

Further test runs of `ptw` with the amended version of `constants.input` are now necessary. These successfully find periodic travelling waves at all points above the Hopf bifurcation locus in the control parameter – wave speed plane. Therefore one can move on to a systematic loop across the plane, checking for wave existence. The number of grid points to consider was set (in `parameter_range.input`) in Stage 1. Thus one can proceed immediately, via the command

```
ptw_loop workedex
```

which completes the loop without any errors; there are a number of warnings about small numbers of continuation steps, but none of these is significant. Before plotting, it is helpful to recreate the Hopf bifurcation locus shown in Figure 2.12 for this new pcode value, using the command

```
hopf_locus workedex 102 1 0 1 6
```

(alternatively the `copy_hopf_loci` command could be used, although this would also copy across the unsuccessful attempt at $\delta = -1$). After this the plotter can be started by typing

```
plot workedex 102
```

and the results of the `ptw_loop` calculation can be shown by typing

```
@pplane
```

at the plotter prompt, selecting the default plot type. To superimpose the Hopf bifurcation locus, one types

```
@hopf_locus 101
```

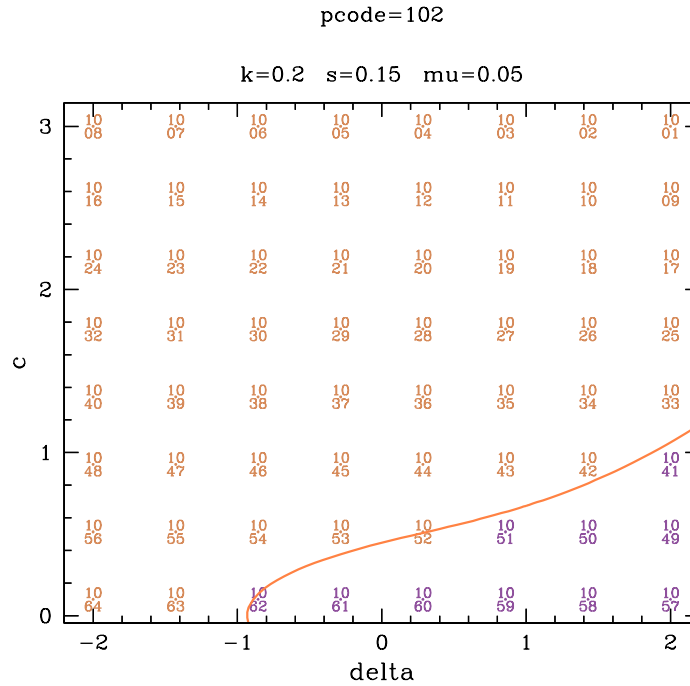


Figure 2.13: Results on the existence of periodic travelling wave solutions for the problem `workedex`. A key illustrating the meaning of the various colours is shown in Figure 2.4 on page 18. The run and plot commands used to generate this figure are given in the main text, and are also listed in the Appendix.

giving the plot illustrated in Figure 2.13. This shows that there are periodic travelling waves at all points in the control parameter–wave speed plane above and to the left of the Hopf bifurcation locus. Although it is not necessary, one might be interested to see the form of one or more of these waves. For example, typing

`@ptw 1037`

gives a plot of the solution with rcode value 1037, which is shown in Figure 2.14. The point in the δ – c plane to which this corresponds can be seen in Figure 2.13. One can now exit the plotter by typing `@exit` or `@quit`.

2.3.9 Stage 9 (Run): Choose `nmesh2` for Stability Calculations

Having determined the part of the δ – c plane in which periodic travelling waves exist, one can move on to a determination of their stability. The first step in this is to determine a suitable value of `nmesh2`. This constant determines the discretisation used for the periodic travelling wave solutions, and choosing a suitable value is often the most difficult part

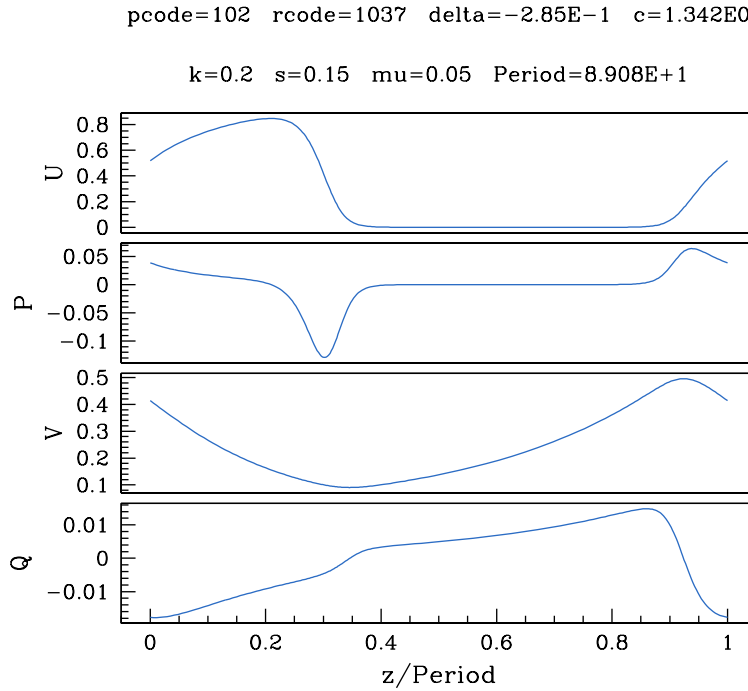


Figure 2.14: An example of a periodic travelling wave solution for the problem `workedx`. The run and plot commands used to generate this figure are given in the main text, and are also listed in the Appendix. Note that different tickmark spacings appear on both the horizontal and vertical axes if `gnuplot` is used as the plotter.

of a WAVETRAIN study. The results of previous stages show that the value `nmesh2=50`, which was inherited from the `template` directory, is large enough for the calculation of periodic travelling waves. But for studying wave stability, the discretisation using `nmesh2` is used to formulate an approximate matrix eigenvalue problem for the eigenfunctions that are periodic over one period of the wave (see §2.1.2). In many cases, it is necessary to increase `nmesh2` in order that these matrix eigenvalues and eigenvectors are a sufficiently good approximation to the actual eigenvalues and eigenfunctions.

To investigate this, one uses the command `eigenvalue_convergence`, which was discussed in §2.1.2. It is necessary to choose a few pairs of control parameter and wave speed values at which to run this command; the four pairs $(\delta, c) = (1.5, 2.0)$, $(-1.0, 2.0)$, $(-1.5, 0.1)$, $(-1.5, 3.0)$ are sensible choices. A selection of values of `nmesh2` must also be chosen. For most problems suitable choices would be the three values 50, 100, 200 or the four values 50, 100, 200, 400. Using the second of these for a more complete study, one runs the four commands

```
eigenvalue_convergence workedex 1.5 2.0 50 100 200 400
eigenvalue_convergence workedex -1.0 2.0 50 100 200 400
eigenvalue_convergence workedex -1.5 0.1 50 100 200 400
eigenvalue_convergence workedex -1.5 3.0 50 100 200 400
```

each of which takes about 30 minutes on a typical desktop computer. The results of each of these runs should be examined for each of the 8 eigenvalues under study; here 8 is the value of `nevalues`, set in `constants.input`. Figure 2.15a shows the results for the fourth eigenvalue, which are typical. Convergence is very rapid as `nmesh2` is increased for $(\delta, c) = (1.5, 2.0)$ and $(-1.5, 0.1)$. For $(\delta, c) = (-1.0, 2.0)$ convergence is slower but `nmesh2=200` appears to give a good approximation to the eigenvalue. However for $(\delta, c) = (-1.5, 3.0)$ there is no obvious convergence pattern. To investigate this further, one can add the calculations for `nmesh2=800`, for this parameter pair only, via the command

```
eigenvalue_convergence workedex -1.5 3.0 800
```

(run time about 2.5 hours on a typical desktop computer). The results for the fourth eigenvalue, which are again typical, are shown in Figure 2.15b; they confirm that the eigenvalues are indeed converging, but much more slowly than for the other control parameter – wave speed pairs considered.

2.3.10 Stage 10 (Input): Change order in `constants.input`

The command “`set_worked_example_inputs 10`” performs the steps described in the previous input stages of the study, and then performs the steps described in this stage.

The results discussed in Stage 9 suggest that the value `nmesh2=400` would be suitable for an investigation of wave stability, and one could proceed on this basis. However, such a large value of `nmesh2` will lead to rather long run times, and it is therefore worthwhile investigating whether a smaller `nmesh2` is possible with a larger value of `order`. This constant is set in `constants.input`, and determines the order of the finite difference approximation used when discretising derivatives in the partial differential equations. WAVETRAIN calculates these finite difference approximations using the algorithm of Fornberg (1998). For (2.1), `order = 1` results in a three-point approximation to the second derivatives, while `order = 2` results in a five-point approximation. Line 12 of `constants.input` should now be changed to:

```
2 # order: accuracy order of approx of highest spatial derivative
```

2.3.11 Stage 11 (Run): Further Tests of Eigenvalue Convergence

Having reset `order`, it is necessary to repeat the runs of `eigenvalue_convergence` for the four selected pairs of control parameter and wave speed values:

```
eigenvalue_convergence workedex 1.5 2.0 50 100 200 400
eigenvalue_convergence workedex -1.0 2.0 50 100 200 400
eigenvalue_convergence workedex -1.5 0.1 50 100 200 400
eigenvalue_convergence workedex -1.5 3.0 50 100 200 400
```

(a)

$\delta = 1.5, c = 2.0$

nmesh	Re of eval	Im of eval	Error bound
50	-0.38794808E-01	0.36312786E+00	0.11995450E-12
100	-0.38928876E-01	0.36369116E+00	0.49653528E-12
200	-0.38962520E-01	0.36383359E+00	0.20041221E-11
400	-0.38971090E-01	0.36387089E+00	0.80468395E-11

$\delta = -1.0, c = 2.0$

nmesh	Re of eval	Im of eval	Error bound
50	0.10404277E+00	0.18641958E+00	0.46635140E-09
100	-0.26550389E-02	0.14226996E+00	0.39632024E-08
200	-0.89938986E-02	0.21837055E+00	0.17400515E-07
400	-0.90764280E-02	0.21810084E+00	0.70767272E-07

$\delta = -1.5, c = 0.1$

nmesh	Re of eval	Im of eval	Error bound
50	0.28543135E-01	0.70237573E-01	0.26369839E-12
100	0.28393371E-01	0.70401889E-01	0.10730430E-11
200	0.28356557E-01	0.70442488E-01	0.42990790E-11
400	0.28347372E-01	0.70452625E-01	0.17200412E-10

$\delta = -1.5, c = 3.0$

nmesh	Re of eval	Im of eval	Error bound
50	0.29021216E+00	0.47746546E+00	0.23805573E-09
100	0.17510311E+00	0.18398707E+01	0.16541475E-09
200	0.31235899E-01	0.22950429E+01	0.89481154E-08
400	-0.78209654E-02	0.21269384E+00	0.19689624E-06

(b)

$\delta = -1.5, c = 3.0$

nmesh	Re of eval	Im of eval	Error bound
800	-0.78641212E-02	0.21268068E+00	0.79270306E-06

Figure 2.15: The change in the eigenvalues of the discretised eigenfunction equations as the constant `nmesh2` is varied, with the constant `order` = 1. These two constants are defined in `constants.input`. For (a), the run commands for the four tables are listed in §2.3.9. Assuming that these are the first runs of the `eigenvalue_convergence` command for `workedex`, they will be allocated ecodes of 101–104. The tables shown can then be generated via the commands `convergence_table workedex 101 4`, `convergence_table workedex 102 4`, `convergence_table workedex 103 4`, `convergence_table workedex 104 4`. For (b), the run command is `eigenvalue_convergence workedex -1.5 3.0 800`. Assuming that the runs for (a) have been done previously, this run will be allocated an ecodes of 105. The table shown can then be generated via the command `convergence_table workedex 105 4`.

$\delta = 1.5, c = 2.0$

nmesh	Re of eval 4	Im of eval 4	Error bound 4
50	-0.38973463E-01	0.36388020E+00	0.15821478E-12
100	-0.38973949E-01	0.36388320E+00	0.65850704E-12
200	-0.38973974E-01	0.36388341E+00	0.26704991E-11
400	-0.38973975E-01	0.36388342E+00	0.10726299E-10

$\delta = -1.0, c = 2.0$

nmesh	Re of eval 4	Im of eval 4	Error bound 4
50	0.83815416E-02	0.77199559E-01	0.11050375E-07
100	-0.91745449E-02	0.21793877E+00	0.52491508E-08
200	-0.91192530E-02	0.21801502E+00	0.23119089E-07
400	-0.91154355E-02	0.21801977E+00	0.94279284E-07

$\delta = -1.5, c = 0.1$

nmesh	Re of eval 4	Im of eval 4	Error bound 4
50	0.28345145E-01	0.70455251E-01	0.34973635E-12
100	0.28344364E-01	0.70455958E-01	0.14289575E-11
200	0.28344317E-01	0.70456000E-01	0.57298591E-11
400	0.28344314E-01	0.70456003E-01	0.22932161E-10

$\delta = -1.5, c = 3.0$

nmesh	Re of eval 4	Im of eval 4	Error bound 4
50	0.14335253E+00	0.54681288E+00	0.40620272E-07
100	0.79166525E-02	0.10178125E+01	0.12918993E-05
200	-0.78877344E-02	0.21267231E+00	0.66805176E-07
400	-0.78808884E-02	0.21267797E+00	0.26276368E-06

Figure 2.16: The change in the eigenvalues of the discretised eigenfunction equations as the constant `nmesh2` is varied, with the constant `order` = 2. These two constants are defined in `constants.input`. The run commands for the four tables are listed in §2.3.11. Assuming that the runs for Figure 2.15 have been done previously, these new runs will be allocated ecode values of 106–109. The tables shown can then be generated via the commands `convergence_table workedex 106 4`, `convergence_table workedex 107 4`, `convergence_table workedex 108 4`, `convergence_table workedex 109 4`.

each of which again takes about 30 minutes on a typical desktop computer. Figure 2.16 shows the results for the fourth eigenvalue, and again the other eigenvalues show a similar convergence pattern. Comparing these results with those in Figure 2.15a shows that the convergence is significantly enhanced by the increase in `order`, enabling a smaller value of `nmesh2` to be used and hence reducing run times.

2.3.12 Stage 12 (Input): Change `nmesh1` and `nmesh2` in `constants.input`

The command “`set_worked_example_inputs 12`” performs the steps described in the previous input stages of the study, and then performs the steps described in this stage.

On the basis of the results described in §2.3.11, the value of `nmesh2` can now be set to 200, and `nmesh1` must also be increased to the same value. Both of these constants are set in `constants.input`.

2.3.13 Stage 13 (Run): Test Runs of the `stability` Command

Having chosen a value of `nmesh2`, one can move on to do some test studies of wave stability. For this, it is convenient to use the same four test pairs of δ and c as in Stages 9 and 11. Therefore one begins by running the four commands

```
stability workedex 1.5 2.0
stability workedex -1.0 2.0
stability workedex -1.5 0.1
stability workedex -1.5 3.0
```

which take only a few minutes each to run on a typical desktop computer. The first and third of these commands run successfully. However the second and fourth have numerical convergence failures, which are associated with the transition from the matrix eigenvalues and eigenvectors to the functional eigenvalues and eigenfunctions. Such convergence failures are often caused by the value of `nmesh2` being too small. However in this case the results obtained in Stage 11 give a reasonable level of confidence in the suitability of `nmesh2`, and thus the likely cause is that the value of `nmesh3` is too small.

2.3.14 Stage 14 (Input): Change `nmesh3` in `constants.input`

The command “`set_worked_example_inputs 14`” performs the steps described in the previous input stages of the study, and then performs the steps described in this stage.

The value of `nmesh3` must now be increased. This constant is set in `constants.input`. Its current value of 50 is inherited from the `template` input subdirectory, and 100 seems a sensible choice as a larger value.

2.3.15 Stage 15 (Run): Further Test Runs of `stability`, and a Run of `stability_loop`

The test runs of the `stability` command done in Stage 13 should now be repeated with the larger value of `nmesh3`; they all run successfully. It may be of interest to visually inspect the spectra for a number of test cases: one does this by typing

```
plot workedex
```

and then running the plot command `@spectrum` for an appropriate set of `r`code values.

One can now move on to a loop over the δ – c plane, investigating wave stability. One does this via the command

stability_loop workedex

(run time: about 4 hours on a typical desktop computer) which loops over all 64 values in the parameter grid that is specified in `parameter_range.input`. This command runs more or less successfully, but reports 6 possible errors arising from the `iposim` being set to 1 rather than 0, and also 3 convergence failures. (Also 9 warnings are issued, about small numbers of continuation steps; these are not a cause for concern.)

A detailed discussion of the parameter `iposim` and the potential causes of the “possible error” messages is given in 3.1.8. Briefly, when `iposim=1` WAVETRAIN only uses matrix eigenvalues with positive imaginary part as starting points for continuation of the spectrum, and this can in some cases lead to erroneous conclusions about stability. To investigate the 6 cases for which WAVETRAIN thinks that the value of `iposim` might be a problem, one plots the spectra for the `rcode` values concerned. In all 6 cases this shows that WAVETRAIN’s conclusion about stability is correct. If there had been any doubt about this then the remedy would have been very simple: change `iposim` to 0 in `constants.input` and run the `stability` command for the δ - c pairs concerned.

As explained in the screen output, details of the convergence failures are given in the file `combined_failures.list` in the directory `output_files/workedex/rcode103/`. Examination of this file shows that all 3 convergence failures occurred for $\delta = -2$, $c = 3$, which has an `rcode` value of 1008: evidently this corner of the parameter plane is a particularly difficult case. The file `output_files/workedex/rcode103/rcode1008/info.txt` contains more detailed information, and shows that for these parameter values WAVETRAIN has been unable to make the transition from the matrix eigenvalues and eigenvectors to the functional eigenvalues and eigenfunctions, for the first, fifth and seventh eigenvalues (ordered by the size of their real part).

Valuable insight into these convergence failures can be obtained by plotting the eigenvalue spectrum that has been calculated by WAVETRAIN in this case. One starts the plotter by typing

```
plot workedex 103
```

(103 is the `rcode` value for the run of `stability_loop`) and then types

```
@spectrum 1008
```

at the plotting prompt. Selecting the default axes limits (and the default rescaling if `gnuplot` is used as the plotter) gives the plot shown in Figure 2.17a. Note the open squares, which indicate the matrix eigenvalues for which there has been a convergence failure. One can focus in on the computed spectrum by rerunning

```
@spectrum 1008
```

and entering

```
-0.036 0.002 -0.05 0.4
```

as the axes limits, giving the plot shown in Figure 2.17b. (If `gnuplot` is used as the plotter, the default rescaling is appropriate.) There is no obvious problem with the computed spectrum, and these figures suggest that the 3 matrix eigenvalues giving convergence failures are artefacts of the discretisation. The plotter can now be exited by typing `@exit` or `@quit`.

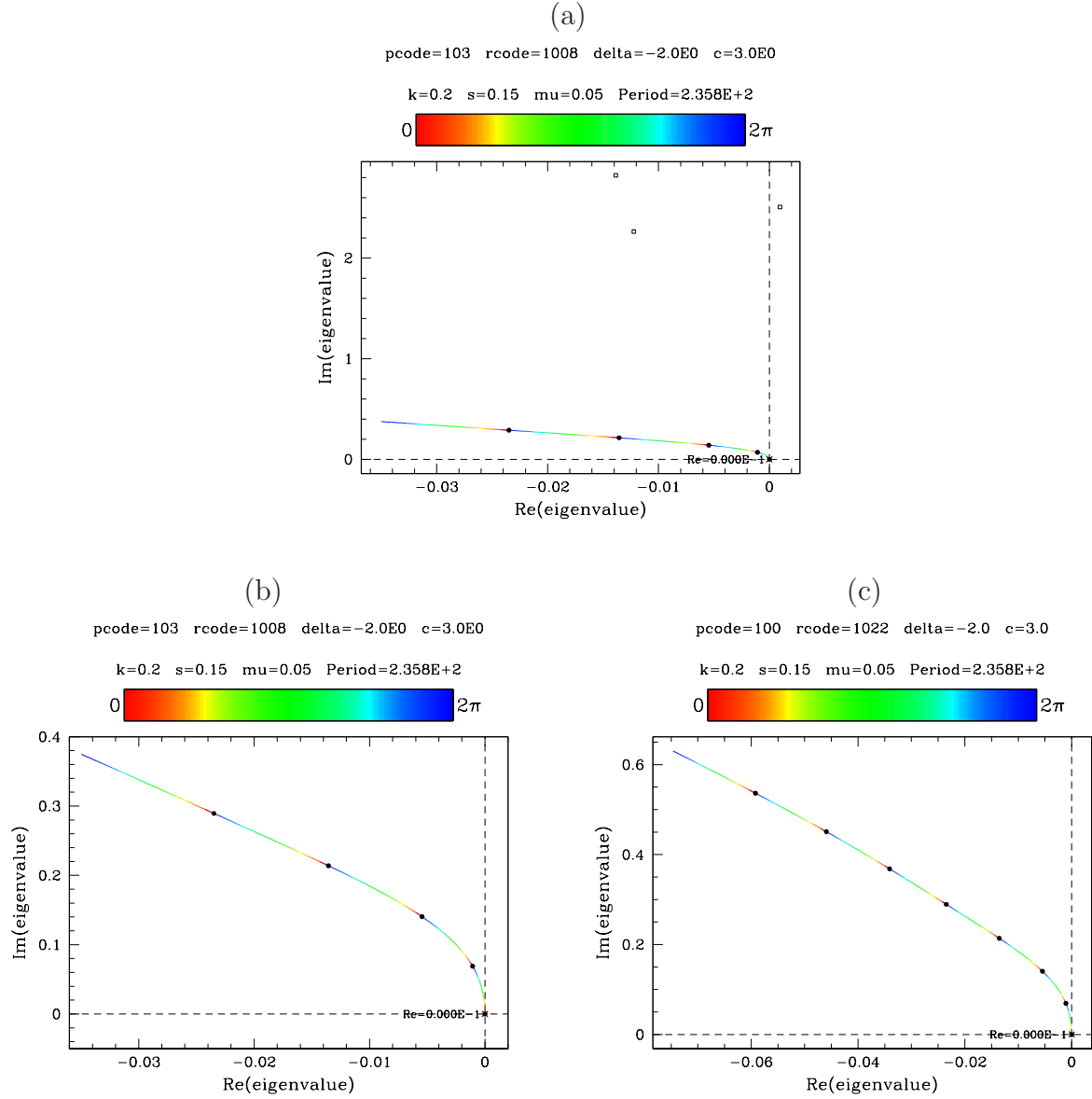


Figure 2.17: Plots of the calculated eigenvalue spectrum for $\delta = -2$, $c = 3$. (a) The plot given by the `@spectrum` command with default axes limits. The open squares denote matrix eigenvalues that gave numerical convergence failure when used as a starting point for calculating the spectrum; the closed circles denote matrix eigenvalues from which spectrum continuation was successful. The value of `nmesh2` was 200 in this case. (b) The same results as (a), but with manually input axes limits. (c) The calculated spectrum with `nmesh2`=400. There are no numerical convergence failures in this case, and default axes limits were used in the plot. The run and plot commands used to generate these figures are given in the main text, and are also listed in the Appendix. Note that if plotting is done using `gnuplot` then ticmarks will be absent from the horizontal axis in these plots; this applies if `spectrumcolour` is set to `rainbow` or `greyscale` or `grayscale`, but not if a single colour is used.

2.3.16 Stage 16 (Input): Change `nmesh1` and `nmesh2` in `constants.input`

The command “`set_worked_example_inputs 16`” performs the steps described in the previous input stages of the study, and then performs the steps described in this stage.

To confirm that the convergence failures for $\delta = -2$, $c = 3$ arise from eigenvalues that are artefacts of the discretisation, one must redo the stability calculation, for these parameters only, with a larger value of `nmesh2`. A sensible choice is to double `nmesh2`, to 400; `nmesh1` must be increased also since it is constrained to be \geq `nmesh2`.

2.3.17 Stage 17 (Run): Run of stability for $\delta = -2$, $c = 3$

With the larger value of `nmesh2`, one should now run the command

```
stability workedex -2.0 3.0
```

(run time: about 30 minutes on a typical desktop computer). The number of previous runs of either `ptw` or `stability` will depend on the number of tests that have been run in previous stages, and this will affect the `rcode` value allocated to this new run, but 1022 would be a typical `rcode` value. The run is successful, with no convergence failures. To view the computed spectrum, one starts the plotter via

```
plot workedex
```

and then enters the command

```
@spectrum 1022
```

(or a different `rcode` value if appropriate), selecting the default axis limits, and the default rescaling if `gnuplot` is used as the plotter. The resulting plot is shown in Figure 2.17c. It has the same appearance as the spectrum calculated with `nmesh2=200` and shown in Figure 2.17b, except that it extends to more negative values of the real part of the eigenvalue; this is because three additional “genuine” eigenvalues have replaced those that were artefacts of the discretisation. The plotter can now be exited by typing `@exit` or `@quit`.

When some of the matrix eigenvalues give convergence failures, WAVETRAIN assesses the wave as stable or unstable based on the part of the eigenvalue spectrum that was calculated using the other matrix eigenvalues. Thus when WAVETRAIN studied the case $\delta = -2$, $c = 3$ as part of the parameter loop in Stage 15, it assessed the wave as being stable. The calculations in this section confirm that this assessment was correct.

2.3.18 Stage 18 (Input): Reset `nmesh1` and `nmesh2` in `constants.input`

The command “`set_worked_example_inputs 18`” performs the steps described in the previous input stages of the study, and then performs the steps described in this stage.

Before proceeding, it is necessary to reset `nmesh1` and `nmesh2` to 200.

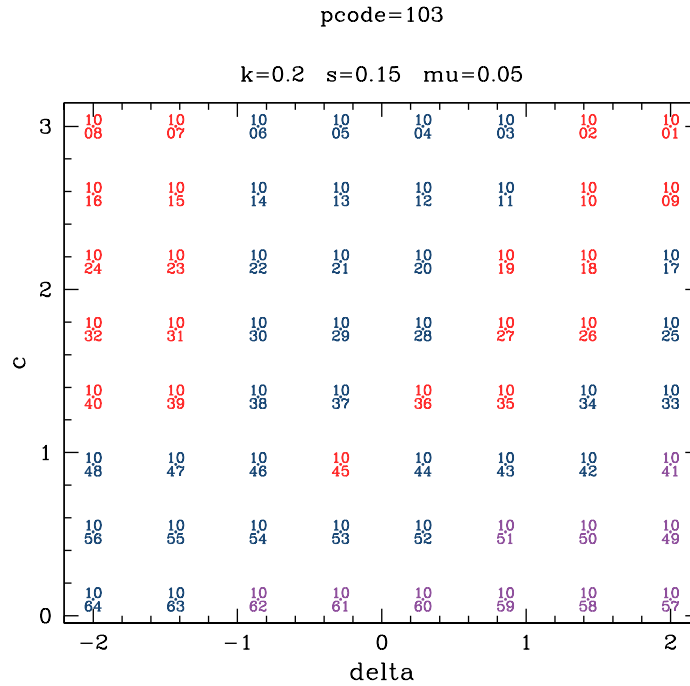


Figure 2.18: Results on the stability of periodic travelling wave solutions for the problem `workedex`. A key illustrating the meaning of the various colours is shown in Figure 2.4 on page 18. The run and plot commands used to generate these figures are given in the main text, and are also listed in the Appendix.

2.3.19 Stage 19 (Run): Plot the Results of the `stability_loop` Run

Having clarified the situation for $\delta = -2$, $c = 3$, one can proceed with plotting the results of the `stability_loop` run done in Stage 15. One starts the plotter by typing

```
plot workedex 103
```

(103 is the relevant pcode value) and then types

```
@pplane
```

at the plotting prompt. Selecting the default plot type (rcode numbers) gives the plot shown in Figure 2.18. There are two separate parameter regions giving stable waves, separated by a region in which waves are unstable. The plotter can now be exited by typing `@exit` or `@quit`.

2.3.20 Stage 20 (Run): Run of `stability_boundary`

The next stage of the study is to trace the boundaries in the δ - c plane between the regions of stable and unstable periodic travelling waves. It is clear from Figure 2.18 that there are

two such boundaries. The starting point for calculating them are pairs of points in the δ - c plane lying on either side. For the right hand boundary, suitable points correspond to rcodes 1019 and 1020, which correspond to $c = 2.171$ with $\delta = 0.857$ and $\delta = 0.285$ respectively. Therefore one types

```
stability_boundary workedex 103 0.857 2.171 0.285 2.171
```

(run time: about 15 minutes on a typical desktop computer). For the left hand boundary, rcode values 1022 and 1023 lie either side; these have $c = 2.171$ with $\delta = -0.857$ and $\delta = -1.428$ respectively. Therefore one types

```
stability_boundary workedex 103 -0.857 2.171 -1.428 2.171
```

(run time: about 15 minutes on a typical desktop computer).

To visualise these stability boundaries, one must first restart the plotter via

```
plot workedex 103
```

and replot the parameter plane via

```
@pplane
```

entering `symbol` as the plot type for variety. One then adds the stability boundaries via the command

```
@stability_boundary all
```

giving the plot shown in Figure 2.19. Although the commands have run successfully, both stability boundary curves in Figure 2.19 are rather jagged in appearance. This indicates that the continuation step sizes are somewhat too large.

2.3.21 Stage 21 (Input): Reduce the Step Sizes for `stability_boundary`

The command “`set_worked_example_inputs 21`” performs the steps described in the previous input stages of the study, and then performs the steps described in this stage.

A factor of 10 should be ample as a reduction in the step size settings for stability boundary calculations. Therefore line 9 of `constants.input` should be changed to:

```
0.01 0.005 0.05 # ds(>0),dsmin,dsmx for stability boundary calcs
```

2.3.22 Stage 22 (Run): Rerun of `stability_boundary`

The two commands

```
stability_boundary workedex 103 0.857 2.171 0.285 2.171
stability_boundary workedex 103 -0.857 2.171 -1.428 2.171
```

should now be rerun. The run time for each is about 30 minutes on a typical desktop computer. Plotting these new stability boundaries (see Figure 2.21 below) confirms that they have a smooth appearance.

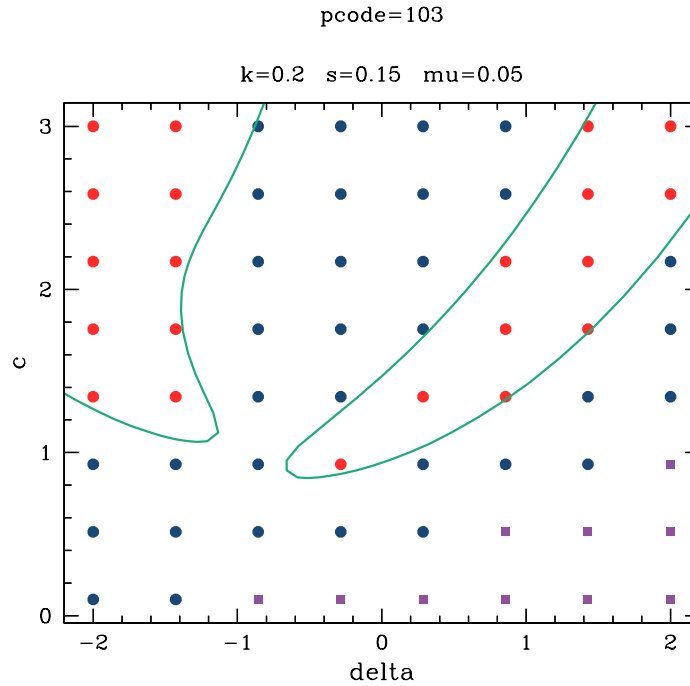


Figure 2.19: Results on the existence of periodic travelling wave solutions, including the boundaries between parameter regions giving stable and unstable waves, for the problem **workex**. The stability boundaries are visibly jagged, due to the continuation step sizes being too large. A key illustrating the meaning of the various colours is shown in Figure 2.4 on page 18. The run and plot commands used to generate this figure are given in the main text, and are also listed in the Appendix.

2.3.23 Stage 23 (Run): Calculate Contours of Constant Period

There is no need to calculate contours of constant period, but it can be helpful in understanding the periodic travelling wave family. To determine suitable contour levels, it is helpful to replot the δ - c plane, showing wave periods rather than rcode values or symbols. To do this one types

@pplane

at the plotter prompt, entering **period** for the plot type. The resulting plot is now shown in Figure 2.20. This shows that 50, 100, 150 and 200 are suitable contour levels. In each case, the **period_contour** command requires two points in the δ - c plane lying either side of the contour, and these can easily be read off Figure 2.20. Having exited the plotter via **@exit** or **@quit**, one can then run the commands to calculate the period contours, with suitable commands being

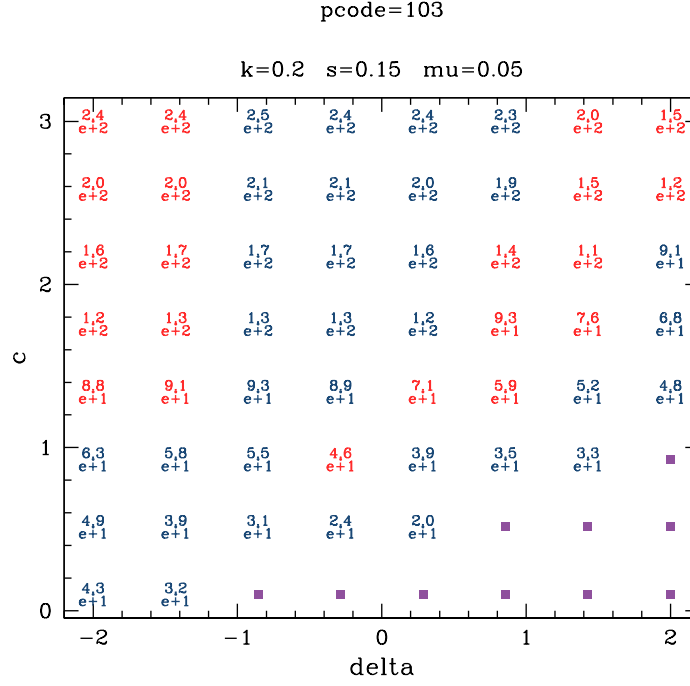


Figure 2.20: Results on the stability of periodic travelling wave solutions for the problem `workdex`, with results displayed using the period of the waves. A key illustrating the meaning of the various colours is shown in Figure 2.4 on page 18. The run and plot commands used to generate this figure are given in the main text, and are also listed in the Appendix.

```

period_contour workedex 103 period=50 -1.0 0.9 -0.2 0.9
period_contour workedex 103 period=100 0.85 1.6 0.85 2.2
period_contour workedex 103 period=150 0.9 2.2 0.2 2.2
period_contour workedex 103 period=200 1.0 2.6 -0.4 2.6

```

which calculate the various contours.

2.3.24 Stage 24 (Input): Copy PDE Simulation Data File

The command “`set_worked_example_inputs 24`” performs the steps described in the previous input stages of the study, and then performs the steps described in this stage.

Before plotting the results of the period contour calculations, it is instructive to add one further ingredient to the plot. Smith & Sherratt (2007) present results from numerical simulations of the partial differential equations (2.1) on a large domain with zero Dirichlet boundary conditions, for a range of values of δ , with $k = 0.2$, $s = 0.15$ and $\mu = 0.05$ as in this worked example. In each case periodic travelling waves develop. Their results

are reproduced in the files `documentation/worked_example/stage24/pde_stable.data` and `documentation/worked_example/stage24/pde_unstable.data`. These files contain two columns, which are the δ values and the measured values of the wave speed, with the two files corresponding to values of δ for which there was not or was visible evidence of instability in the waves. The data in these files can be incorporated into WAVETRAIN plots via the plot command `@pointsfile` (see page 127), and the files should now be copied into the `workdex` input subdirectory.

2.3.25 Stage 25 (Run): Final Plots

The various results can now be gathered together in a final plot. Firstly, in order to include the locus of Hopf bifurcation points in this plot it is necessary to recalculate it for pcode 103, via the command

```
hopf_locus workedex 103 1 0 1 6
```

(alternatively one could use the `copy_hopf_loci` command, although WAVETRAIN would issue queries because of the differences in the input files between pcode 103 and pcodes 101/102). One then starts the plotter via

```
plot workedex 103
```

and enters the plotter command

```
@pplane
```

entering `symbol` as the plot type, which is most suitable for a final plot. One then enters the commands

```
@hopf_locus 101
@stability_boundary 103
@stability_boundary 104
```

to add the locus of Hopf bifurcation points and the stability boundaries. Note that 103 and 104 are the scode values for the runs of `stability_boundary` done in Stage 22, with reduced step sizes. To add the period contours, one enters

```
@period_contour all
```

at the plotter prompt. The user is then prompted about the location of contour labels. By default these are placed at the edges of the plot, but in this case a clearer plot is given by entering

```
delta=-1.8 delta=1.55
```

as the label location. Finally the data from the partial differential equation simulations is added to the plot via the two commands

```
@pointsfile "pde_stable.data" 9
@pointsfile "pde_unstable.data" 8
```


which assigns pcode value 104 to the new parameter plane. One then copies the various results across via the commands

```
copy_hopf_loci workedex 103 104
copy_period_contours workedex 103 104
copy_stability_boundaries workedex 103 104
```

and types

```
plot workedex 104
```

to restart the plotter. One then simply repeats the series of plotter commands used to generate Figure 2.21; the only difference is that the `@ppplane` command does not prompt the user for the plot type, since no (δ, c) points are being plotted. The resulting plot is shown in Figure 2.22.

2.3.26 Stage 26 (Run): Delete Unwanted Output Files

WAVETRAIN generates significant amounts of numerical data. To avoid this accumulating, it is good practice to remove unwanted output files at the end of a study. A series of commands are provided for this purpose, and they are described in detail in §3.1.11. For this worked example, all runs associated with pcode values 101 and 102 are now redundant, and the corresponding output files can be deleted via the commands

```
rmrcode workedex 101
rmrcode workedex 102
```

which prompt the user for confirmation. In addition, the data files associated with the large number of runs of `ptw`, `stability` and `eigenvalue_convergence` are all no longer needed. All WAVETRAIN runs such as these, which are not associated with a parameter plane plot, have their output data stored under the notional pcode value 100 (see §3.1.11 and §4.4 for further details). Therefore the output data can be deleted via the command

```
rmrcode workedex 100
```

which issues three separate prompts for confirmation, since this command has the potential to delete a large amount of output data.

2.4 Documentation and Help Facilities

In the main WAVETRAIN directory there is a subdirectory `documentation` containing a PDF copy of this user guide.

A basic help facility is provided in WAVETRAIN via the command `wt_help`; like all commands, the user must be in the main WAVETRAIN directory to use it. Typing

```
wt_help <command>
```

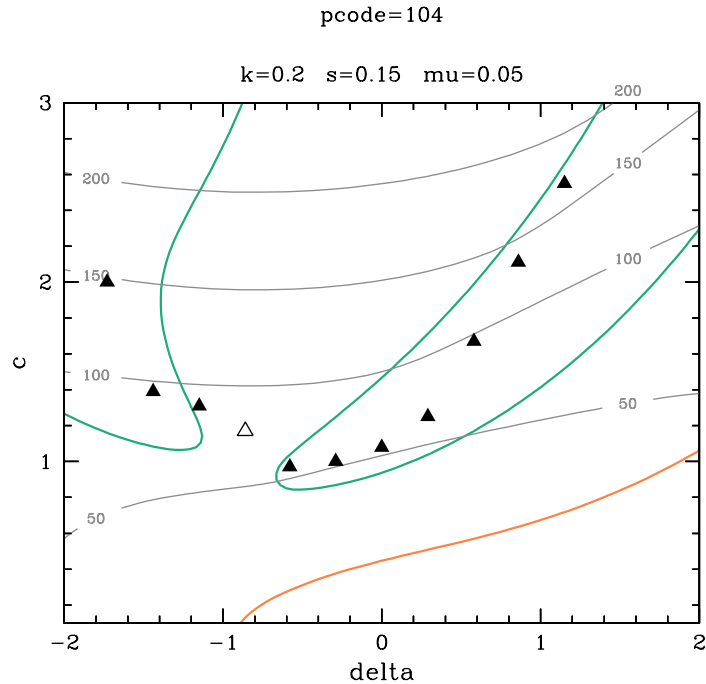


Figure 2.22: An alternative version of the final plot of the control parameter–wave speed plane for the problem `workdex`. This figure is the same as Figure 2.21 except that the symbols indicating the (δ, c) points used in the run of `stability_loop` are omitted. A key illustrating the meaning of the various colours and symbols is shown in Figure 2.4 on page 18, except for the black triangles, which indicate results from numerical simulations of the partial differential equations. Open/closed triangles indicate solutions with/without visible evidence of instability (see main text for details). The run and plot commands used to generate this figure are given in the main text, and are also listed in the Appendix.

provides a brief summary of the usage of the specified command, which can be either a run or plot command. If no argument is given, `wt_help` lists the WAVETRAIN run and plot commands. By default the output from `wt_help` is scrolled on the screen, but this can be changed to a listing of the appropriate help file by editing `defaults.input`.

Help for plotter commands is also available from within the plotter. Typing `@help` followed by a command name gives a summary of the command’s syntax and usage. Typing `@help` alone gives a listing of the plot commands. By default the plot command `@help` scrolls through the corresponding help file, but this can be changed to a listing by changing `helpdisplay` in `plot_defaults.input`.

For the benefit of users wishing to understand the workings of WAVETRAIN, each subdirectory contains a `README` file that summarises the role played by each program within it. There is also a `README` file in the main WAVETRAIN directory, which contains the version name/number.

2.5 Troubleshooting

It is anticipated that most users will run WAVETRAIN commands interactively, with the constant `info` set to 3 (in `constants.input`). This results in limited screen output, charting the overall progress of the run. If a problem occurs, then this screen output may be sufficient to resolve it. Otherwise, or if `info` is set to 1 or 2, users can see much more detailed information on the run in a file called `info.txt`. This file will be located in a subdirectory of `output_files`. Full details of the subdirectory structure of `output_files` are given in §4.4.1, and the following list simply gives the location of `info.txt` files associated with the various WAVETRAIN commands.

bifurcation_diagram This command has not yet been discussed: see §3.1.6. Detailed information on a run of this command is contained in
`output_files/pcode100/bcode<code_number>/info.txt`

eigenvalue_convergence Detailed information on a run of this command is contained in
`output_files/pcode100/ecode<code_number>/info.txt`

hopf_locus Detailed information on a run of this command is contained in
`output_files/pcode<code_number>/ccode<code_number>/info.txt`

period_contour Detailed information on a run of this command is contained in
`output_files/pcode<code_number>/ccode<code_number>/info.txt`

ptw and stability Detailed information on a run of these commands is contained in
`output_files/pcode100/rcode<code_number>/info.txt`

ptw_loop and stability_loop Detailed information on a run of these commands is divided between a number of different `info.txt` files. Details of overall progress through the loop over the grid of control parameter and wave speed values is contained in `output_files/pcode<code_number>/info.txt` and specific information about the run for a given pair of control parameter and wave speed values is contained in `output_files/pcode<code_number>/rcode<code_number>/info.txt`

stability_boundary Detailed information on a run of this command is contained in
`output_files/pcode<code_number>/scode<code_number>/info.txt`

One component of these `info.txt` files is a list of any errors and warnings that have occurred during the run. For convenience, this list is reproduced in a file `errors.txt` in the same subdirectory. (If no errors or warnings have occurred, this file will be present but empty).

During runs of its various commands, WAVETRAIN generates a large number of temporary files (see §4.3.1 for details). By default, these files are deleted at the end of a run, but they can be retained by setting the constant `iclean` to 2 in `constants.input`. In some cases, this can also be useful in attempting to understand the cause of a problem. Once the files have been examined, they can be deleted by the command `cleanup`.

Part 3

Advanced Features of WAVETRAIN

3.1 Further Details of Run Commands

3.1.1 The Format of Numerical Inputs, and Precision

Numerical inputs to WAVETRAIN are required in the various input files, and also as command arguments. They can be supplied in either decimal or scientific notation. Thus 123.4, 1.234E2 and 1.234E+2 are all acceptable. Entries that happen to be integer-valued can also be entered without a decimal point: for example the control parameter value 3.0 can also be entered as 3. However there is one important case in which an integer value and its decimal equivalent are treated differently, namely in powers that appear in `equations.input`. A term such as “`x**3`” in `equations.input` is evaluated as one would expect, by multiplying `x` by itself three times; however “`x**3.0`” is evaluated using the logarithm and exponential functions, which is less accurate. Therefore to preserve accuracy, integer powers should always be entered as integers. Of course, integer variables such as the numbers of grid points in `parameter_range.input` must be given as integers.

WAVETRAIN does its computations using Fortran77 programs, which are written by the package on the basis of the input files. It uses 8-byte (“double”) precision throughout, meaning that there are 16 significant decimal digits. Users who are familiar with Fortran77 will know that this level of precision requires that numerical values are specified explicitly as being double precision. For example 1.0/3.0 gives only eight 3’s after the decimal point, with the remaining digits being random; in Fortran77 one must instead write 1.0D0/3.0D0. WAVETRAIN takes account of this when converting the input files into Fortran77 code, changing E’s to D’s in scientific notation and adding “D0” to decimals and “.0D0” to integers in order to retain double precision accuracy throughout. An important exception to this is integer powers, which are left as integers (see the discussion in the previous paragraph).

A final point about numerical format concerns upper and lower case. According to the international standard for Fortran77 (see <http://gcc.gnu.org/wiki/GFortranStandards> and <http://rsusu1.rnd.runnet.ru/develop/fortran/prof77/index.html>), programs must be written entirely in upper case, and this is true for all code used in WAVETRAIN. However, in principle, this should also apply to numerical inputs. Thus a numerical input such as 1.234e2 is not compatible with the international standard for Fortran77: it

should be 1.234E2. In reality, all modern Fortran77 compilers will accept both lower and upper case, but it is conceivable that a user might have a Fortran77 compiler that gives an error because of entries with a lower case “e”. In this highly unlikely event, the remedy is simple: change to upper case “E” in any numerical values entered using scientific notation.

3.1.2 Optional Arguments of the `stability_boundary` Command

The command `stability_boundary` has six compulsory arguments; the first two are the subdirectory name and pcode value, and arguments 3–6 are the coordinates of two points in the control parameter–wave speed plane lying on either side of the stability boundary. It is also permissible to give additional argument(s). These have one of two possible forms: either `<control_parameter_name> = <value>` or `<wave_speed_name> = <value>`. Here the labels “`<control_parameter_name>`” and “`<wave_speed_name>`” denote the names of the control parameter and the wave speed, as given in `variables.input`. There is no upper limit on the number of such additional arguments, and they cause detection of points at which the stability boundary crosses the specified values; if there are multiple crossings, they are all recorded. These crossings are written to the output file `info.txt`, and also to the screen if the constant `info` (defined in `constants.input`) is 3 or 4. They can be retrieved subsequently via the command `list_crossings`. Thus for the example problem `demo` discussed in the previous section, the command

```
stability_boundary demo 102 1.75 0.3 1.75 0.7 A=0.7 c=0.55 A=1.1 A=2.5 c=0.65
```

causes the stability boundary to be calculated as described in §2.1, but with the additional recording of the points at which the stability boundary crosses $A = 0.7, 1.1, 2.5$ and $c = 0.55, 0.65$. If no subsequent runs have been done other than those listed in §2.1 then this run will be allocated the scode value 102, and the command

```
list_crossings demo 102 102
```

will give a list of the crossing points. (Here the first 102 is the pcode value and the second is the scode value). Note that periodic travelling wave solutions corresponding to the crossing points are not recorded: this is in contrast to the `period_contour` command (see §3.1.3). The crossing information is not used in plotting.

3.1.3 Optional Arguments of the `period_contour` Command

The command `period_contour` has either four or seven compulsory arguments. The first two are the subdirectory name and pcode value. If the period is specified in the third argument, then arguments 4–7 are the coordinates of two points in the control parameter–wave speed plane lying on either side of the period contour; otherwise arguments 3 and 4 are the coordinates of the point through which the period contour is to be drawn. It is also permissible to give additional argument(s). These have one of two possible forms: either `<control_parameter_name> = <value>` or `<wave_speed_name> = <value>`. Here the labels “`<control_parameter_name>`” and “`<wave_speed_name>`” denote the names of the control parameter and the wave speed, as given in `variables.input`. There is

no upper limit on the number of such additional arguments, and they cause detection of points at which the period contour crosses the specified values; if there are multiple crossings, they are all recorded (up to a maximum number of 99). The periodic travelling wave solutions corresponding to the crossing points are also recorded. (Note that this is not the case when optional arguments are used for the `stability_boundary` command: see §3.1.2.) The crossings are listed in the file `pcsolution.list` in the relevant output subdirectory, and this list contains a two-digit code number for each crossing. The number of crossings is limited to 99 and in the unlikely event that the number of crossings exceeds this maximum, only the first 99 to be detected will be recorded. The order of the crossings is simply the order in which they were detected during numerical continuation: therefore if more than one optional argument is given, the code numbers are not grouped according to the corresponding optional argument. For each crossing, the output subdirectory also contains the file `pcsolution<code>.soln` which gives the corresponding periodic travelling wave solution. The first column of the file contains values of the periodic travelling wave coordinate, and the remaining columns contain the corresponding values of the travelling wave variables, in the order in which they are listed in `variables.input`. The solution will span exactly one period, and the travelling wave variables in the first and last rows will be the same. The output subdirectory also contains a one-line file `pcsolution<code>.params` which gives the control parameter and wave speed values corresponding to the solution. The list of crossings can be viewed via the command `list_crossings`. The crossing information is not used in plotting.

3.1.4 The Commands `set_homoclinic` and `unset_homoclinic`

WAVETRAIN does not have the capability to track the loci of actual homoclinic solutions of the travelling wave equations. Rather, these must be approximated by the loci of solutions of large period, as discussed in §2.1. For the purposes of plotting, it is often convenient to represent such a locus differently from that of other contours of constant wave period. Typing `set_homoclinic demo 102 101` (for example) designates the contour with ccode 101 for the parameter plane with pcode 102 to be an (approximation to) a locus of homoclinic solutions. This is simply a relabelling: no new computation is done. Its effect is that the plotter command `@period_contour` plots that contour in a different colour, and omits the label(s) showing the value of the period. The designation can be reversed by typing `unset_homoclinic demo 102 101`. Note that if `set_homoclinic/unset_homoclinic` are applied to a contour that is currently designated as being/not being homoclinic, it simply has no effect. (No error is reported.)

As an example of the use of these commands, consider the plot shown in Figure 2.5 on page 21. This plot was for a run with pcode 101, and shows a control parameter–wave speed plane with two contours of constant wave period, with periods 3000 (ccode 101) and 80 (ccode 102). Typing

```
set_homoclinic demo 101 101
```

designates the first of these to be a locus of homoclinic solutions. Replotting via S

```
plot demo 101
```

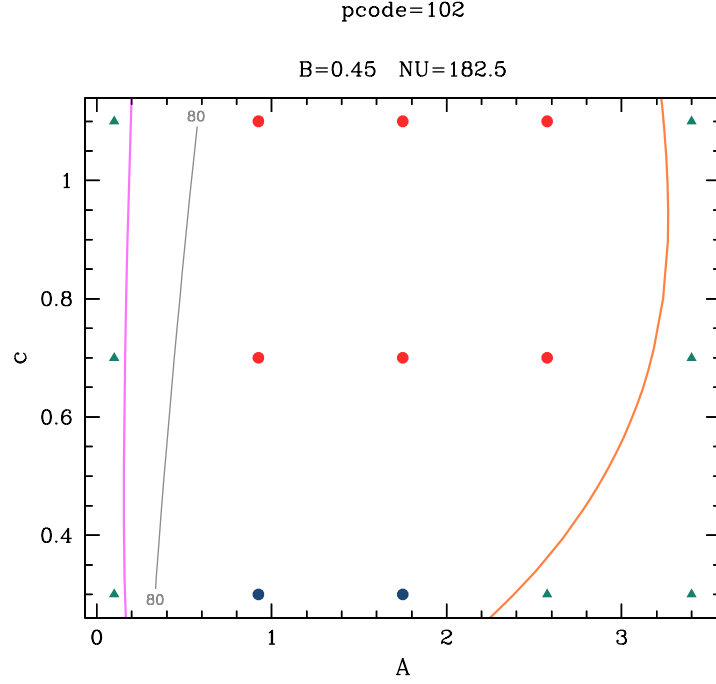


Figure 3.1: An illustration of the use of the run command `set_homoclinic`. The figure shows the Hopf bifurcation locus and two contours of constant wave period for the problem `demo`, one of which has been designated to be a locus of homoclinic solutions. These curves are superimposed on a plot showing periodic travelling wave existence in the control parameter–wave speed plane. A key showing the meaning of the various symbols and colours (including that used for the homoclinic locus) is shown in Figure 2.4 on page 18. The run and plot commands used to generate this figure are given in the main text, and are also listed in the Appendix.

followed by the plot command

```
@pplane
```

(selecting `symbol` as the plot type) and then

```
@hopf_locus 101
```

and

```
@period_contour all
```

(selecting the default label location) gives the plot illustrated in Figure 3.1. The contour of period 3000 is shown in a different colour and weight, and without labels showing the period.

3.1.5 The Command `new_pcode`

Having calculated the parameter plane for demo, as described in §2.1, one may wish to obtain the corresponding plot for a slightly different value of one of the other parameters, for instance $B = 0.4$ rather than $B = 0.45$. (Recall that these parameters are given in the file `other_parameters.input`). One might naturally assume that the basic structure of the plot would be the same, making a scan across the parameter plane unnecessary: one would just need to recompute the locus of Hopf bifurcation points, the “homoclinic” locus, and the stability boundary. To do this one requires a pcode value, which has previously been generated via a run of `ptw_loop` or `stability_loop`. The command `new_pcode` is provided for this situation; it was introduced previously in §2.3. After editing the file `other_parameters.input`, one types

```
new_pcode demo
```

and the new pcode value is reported. The commands `hopf_locus`, `period_contour` and `stability_boundary` can then be used to generate all the data for the parameter plane plot. Note that if the plot command `@pplane` is used for a pcode value created via `new_pcode`, it causes axes, labels and titles to be drawn around a blank plot; this is necessary before using the plot commands `@hopf_locus`, `@period_contour` or `@stability_boundary`.

3.1.6 The Command `bifurcation_diagram`

An important capability of WAVETRAIN that was not discussed in section 2.1 is the generation of bifurcation diagrams. The basic command has a form such as

```
bifurcation_diagram demo c=0.8
```

which generates a bifurcation diagram as the control parameter is varied, for wave speed $c = 0.8$. (The wave speed name `c` is set in the file `variables.input`). The limits on the control parameter for the bifurcation diagram are taken from `parameter_range.input`. The run is allocated a three digit “bcode” number (101-999); since this is the first bifurcation diagram run, the bcode will be 101. The diagram can be plotted by starting the plotter via

```
plot demo
```

(no pcode value is needed) and then running the plotter command

```
@bifurcation_diagram 101
```

(101 is the bcode value). The user is asked to specify whether the vertical axis of the plot should show the L2-norm of the travelling wave solution and/or the L2-norm of the steady state, or the wave period. There is also the opportunity to manually override the default axes limits; this is often necessary for the vertical axis when using the wave period if the plotting region includes a homoclinic solution. Selecting “`norm`” for the vertical axis and default axes limits gives the plot shown in Figure 3.2a. In bifurcation diagram

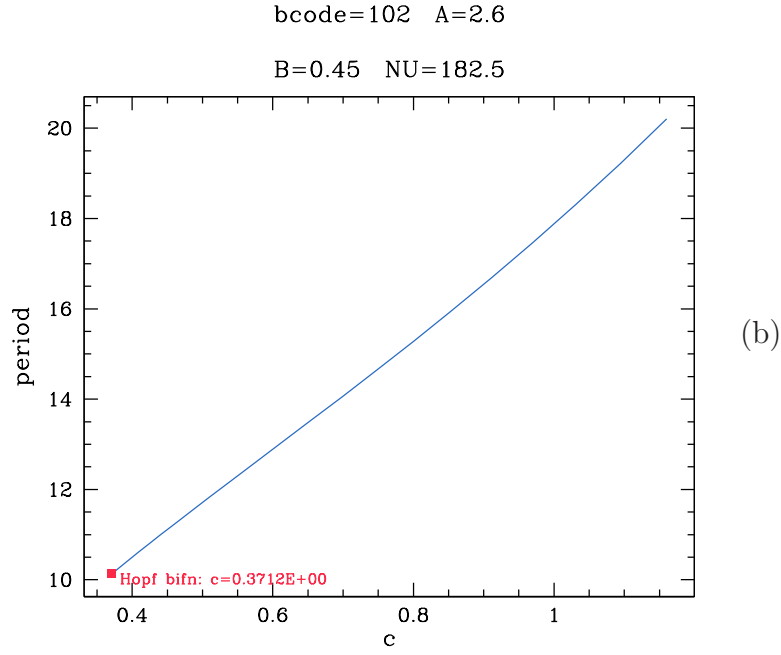
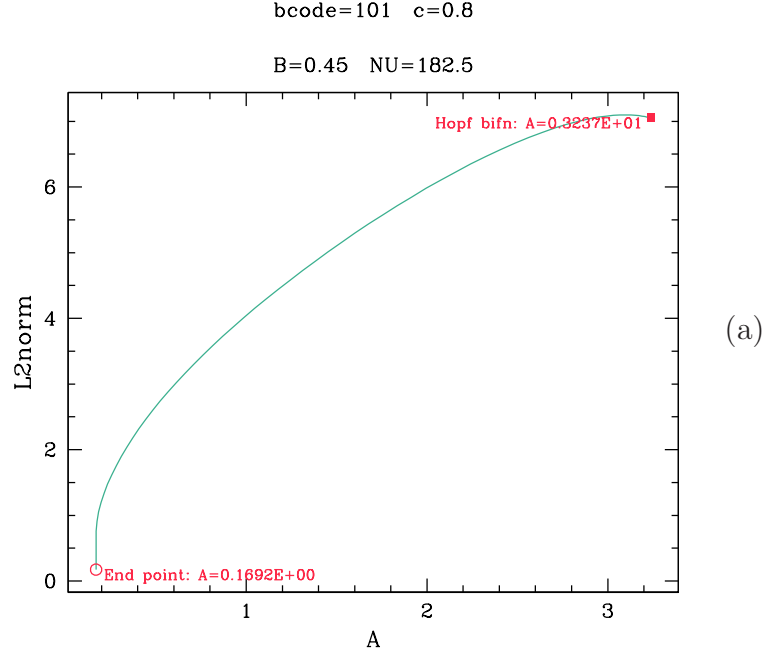


Figure 3.2: Bifurcation diagrams for the problem `demo`. The control parameter and wave speed as used as the bifurcation parameter in (a) and (b) respectively. The L2-norm of the solution is plotted on the vertical axis in (a), and the period is used in (b). The run and plot commands used to generate these figures are given in the main text, and are also listed in the Appendix.

plots, a filled square denotes a Hopf bifurcation point, an open circle denotes the end of a solution branch, and if there are folds in the solution branch then these are indicated by filled circles (see for example Figure 3.6). The labels beside these symbols, and if desired also the symbols themselves, can be suppressed by changing the plotter setting `showbdlabls`, which is defined in `plot_defaults.input`.

As an alternative to specifying a wave speed value and using the control parameter as the bifurcation parameter, the roles can be reversed. For example the command

```
bifurcation_diagram demo A=2.6
```

will generate a bifurcation diagram with the wave speed as the bifurcation parameter; this is illustrated in Figure 3.2b. (The control parameter name `A` is set in the file `variables.input`).

The `bifurcation_diagram` command also has optional additional arguments. One of these is the name of one of the travelling wave variables (as specified in `variables.input`). If this is included then the same computations are performed but with different output data. Rather than recording the L2-norm of the whole travelling wave solution, the maximum, minimum, mean, L2-norm and steady state value of the specified variable are recorded. One can select which of these are plotted when one runs the plot command `@bifurcation_diagram`. An example run command would be

```
bifurcation_diagram demo variable=W c=0.6
```

which will be allocated a bcode value of 103 since it is the third bifurcation diagram to be calculated. Here the order of the second and third arguments does not matter, so that

```
bifurcation_diagram demo c=0.6 variable=W
```

is equivalent. To plot the resulting bifurcation diagram, one starts the plotter by typing

```
plot demo
```

and then gives the plot command

```
@bifurcation_diagram 103
```

(103 is the bcode value). The user will then be prompted to choose from a variety of different options for what is shown on the vertical axis. For example, selecting “`norm max min`” gives the plot shown in Figure 3.3a, while “`max mean stst`” gives the plot shown in Figure 3.3b. In both cases, the plots shown use the default axes limits. Note that the legend boxes are included automatically when there is more than one solution property in the plot.

The other type of optional argument for the `bifurcation_diagram` command is the specification of one or more files as starting points for solution branches. The first stage of the `bifurcation_diagram` command is always to scan the range of either control parameter or wave speed values given in `parameter_range.input`, looking for Hopf bifurcations. The second stage is then to calculate the solution branch emanating from each of these Hopf bifurcation points. If one or more optional arguments `file=<filename>` is given then there is a third stage, in which solution branches are calculated using

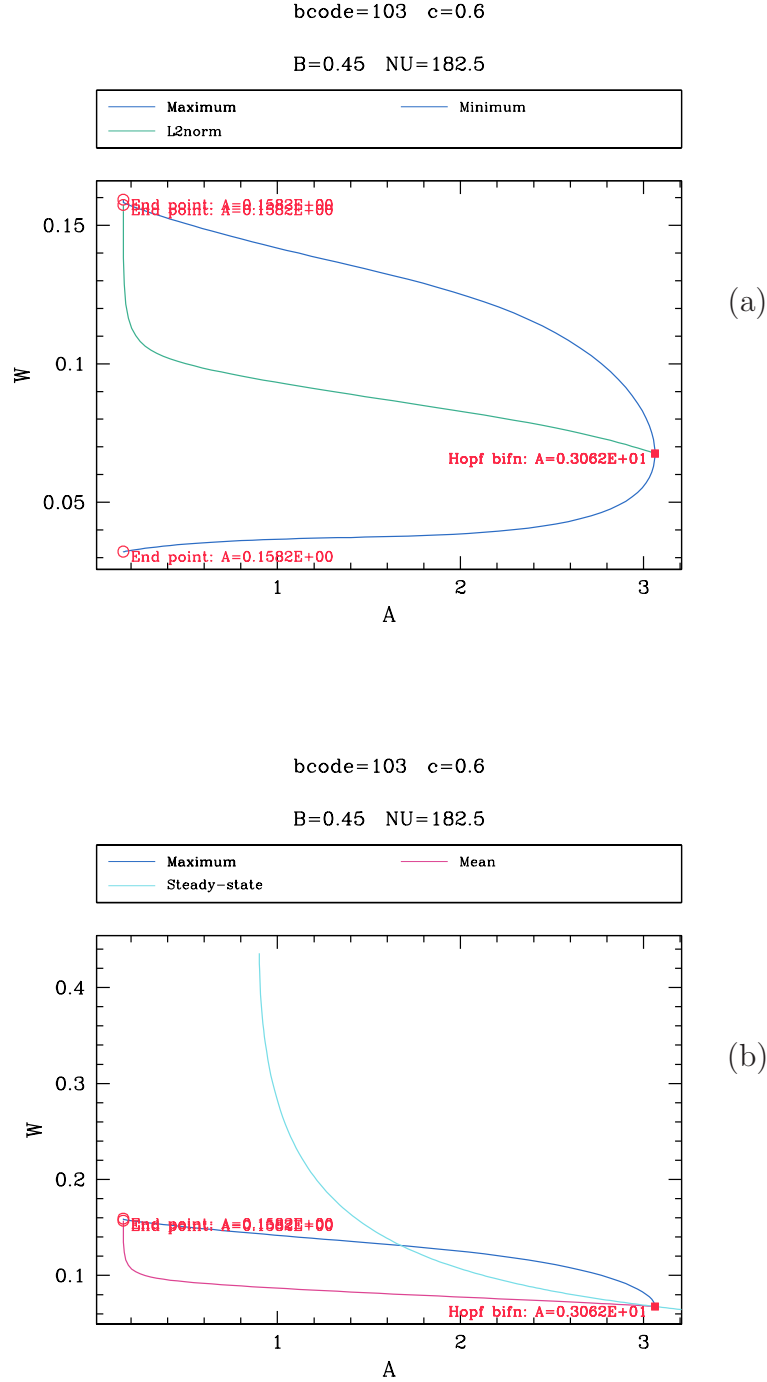


Figure 3.3: Bifurcation diagrams for the problem `demo`, with W specified as the study variable in the command line. Parts (a) and (b) are for the same run, but with different options selected for the vertical axis. Note that the same line colour is used for the maximum and minimum; this can be changed by altering `bdlinecolourmax` and/or `bdlinecolourmin` in the plotter style file (see §3.4.2). The run and plot commands used to generate these figures are given in the main text, and are also listed in the Appendix.

the periodic travelling wave solution `<filename>.soln` as a starting point. The file `<filename>.soln` must be in the appropriate subdirectory of `input_files`, and the required format is:

column 1: travelling wave (or space) coordinate

remaining columns: the travelling wave variables, in the order in which they are listed in `variables.input`.

There are no particular constraints on the discretisation used for the solution in the file. It must be given over one period of the periodic travelling wave, and the travelling wave variable values given in the last line must be exactly the same as those given in the first line. The corresponding space/travelling wave coordinate will therefore be one period greater than that in the first line. A second file `<filename>.params` is also required, again in the `input_files` subdirectory; this is a one-line file containing the control parameter and wave speed values in the first and second columns respectively.

The main motivation for the `file=` optional argument is that it enables the command `bifurcation_diagram` to calculate periodic travelling wave solution branches that begin and end at homoclinic solutions, and are not connected to a Hopf bifurcation point. However its use can be illustrated using the `demo` example. The command

```
bifurcation_diagram demo A=2.0
```

would return an error message saying that no Hopf bifurcation was found as the wave speed c varied between 0.3 and 1.1 (specified in `parameter_range.input`) for $A = 2.0$. In fact there is a periodic travelling wave solution for all wave speeds in this range, but WAVETRAIN cannot calculate the solution branch without a Hopf bifurcation as a starting point. One obvious remedy would be to widen the range of wave speeds being considered. But an alternative is to calculate a periodic travelling wave solution as a starting point. For example, one could use the command

```
ptw demo 2.0 0.8
```

to calculate a suitable starting solution. Following a successful run of either `ptw` or `stability`, the output subdirectory contains files with the correct format for a starting point for `bifurcation_diagram`. These files are named using the control parameter and wave speed, in this case `ptw_2.0_0.8.soln` and `ptw_2.0_0.8.params`. After copying these files into the `demo` subdirectory of `input_files`, one can run the command

```
bifurcation_diagram demo A=2.0 file=ptw_2.0_0.8
```

to calculate the required solution branch.

Any number of `file=<filename>` arguments is permitted; WAVETRAIN will use each file in turn as a starting point. The order of the files does not matter. The options `file=` and `variable=` can be used together, and the order of all arguments beyond the first (the subdirectory name) does not matter.

3.1.7 Keeping Track of WAVETRAIN Runs

If one is doing a large number of different WAVETRAIN runs for the same problem, it can be a challenge to keep track of which run is which. A useful aid in this situation is the file `command.txt`, which is present in each output subdirectory corresponding directly to a WAVETRAIN command. For example, in §2.1.1, the command

```
hopf_locus demo 101 2.0 0.8 3.5 0.8
```

was used to calculate a Hopf bifurcation locus, and was allocated the hcode value 101. Then the file `command.txt` in the directory `output_files/demo/pcode101/hcode101/` contains a single line, in which the command is repeated. Further details of `command.txt` and of other WAVETRAIN output files are given in §4.4.1.

Runs of `ptw_loop` and `stability_loop` create a `command.txt` file in the pcode output directory, but not in the individual rcode subdirectories. To help keep track of the runs in this situation, WAVETRAIN has the command `list_rcodes`. This has one argument, a pcode value, and it lists the control parameter and wave speed associated with each rcode value for that pcode, plus the overall outcome of the calculation, and the period of the wave if one was found. If the argument is either omitted or is equal to 100, then the list corresponds to waves calculated individually, via either the `ptw` or `stability` commands.

3.1.8 The Parameter `iposim`

The spectrum of a periodic travelling wave is symmetric about the real axis. Therefore in order to investigate stability it is unnecessary to calculate the whole of the spectrum: the part in the upper half of the complex plane is sufficient. With this in mind, WAVETRAIN provides the constant `iposim`, which is set in `constants.input`. When WAVETRAIN calculates eigenvalues corresponding to periodic eigenfunctions, it records either all eigenvalues or only those with imaginary part ≥ 0 , according to whether `iposim`=0 or 1. The total number of recorded eigenvalues is specified by `nevals`, which is also set in `constants.input`, and the recorded eigenvalues are used as starting points for numerical continuation of the spectrum. In most cases, setting `iposim`=1 will give all of the spectrum in the upper half plane, and possibly also the parts of some loops which extend into the lower half plane. This part of the spectrum will correctly determine the stability of the periodic travelling wave. The great advantage of restricting the eigenvalues corresponding to periodic eigenfunctions to those with imaginary part ≥ 0 is that typically it almost halves the computational time taken to determine periodic travelling wave stability.

However for some spectra it is necessary to set `iposim`=0 in order to calculate in full the part of the spectrum in the upper half plane. Schematic illustrations of two such cases are given in Figure 3.4. In these sketches the black dots indicate eigenvalues corresponding to periodic eigenfunctions. Consider first the case shown in part (a) of the figure. The spectrum consists of two loops, one of which crosses the imaginary axis, implying that the periodic travelling wave is unstable. However for `iposim`=1 only the eigenvalues with imaginary part ≥ 0 corresponding to periodic eigenfunctions are used as starting points for continuation of the spectrum. Consequently only half of the larger loop is calculated, and the part that crosses into the right hand half plane is missing. Similarly for the case

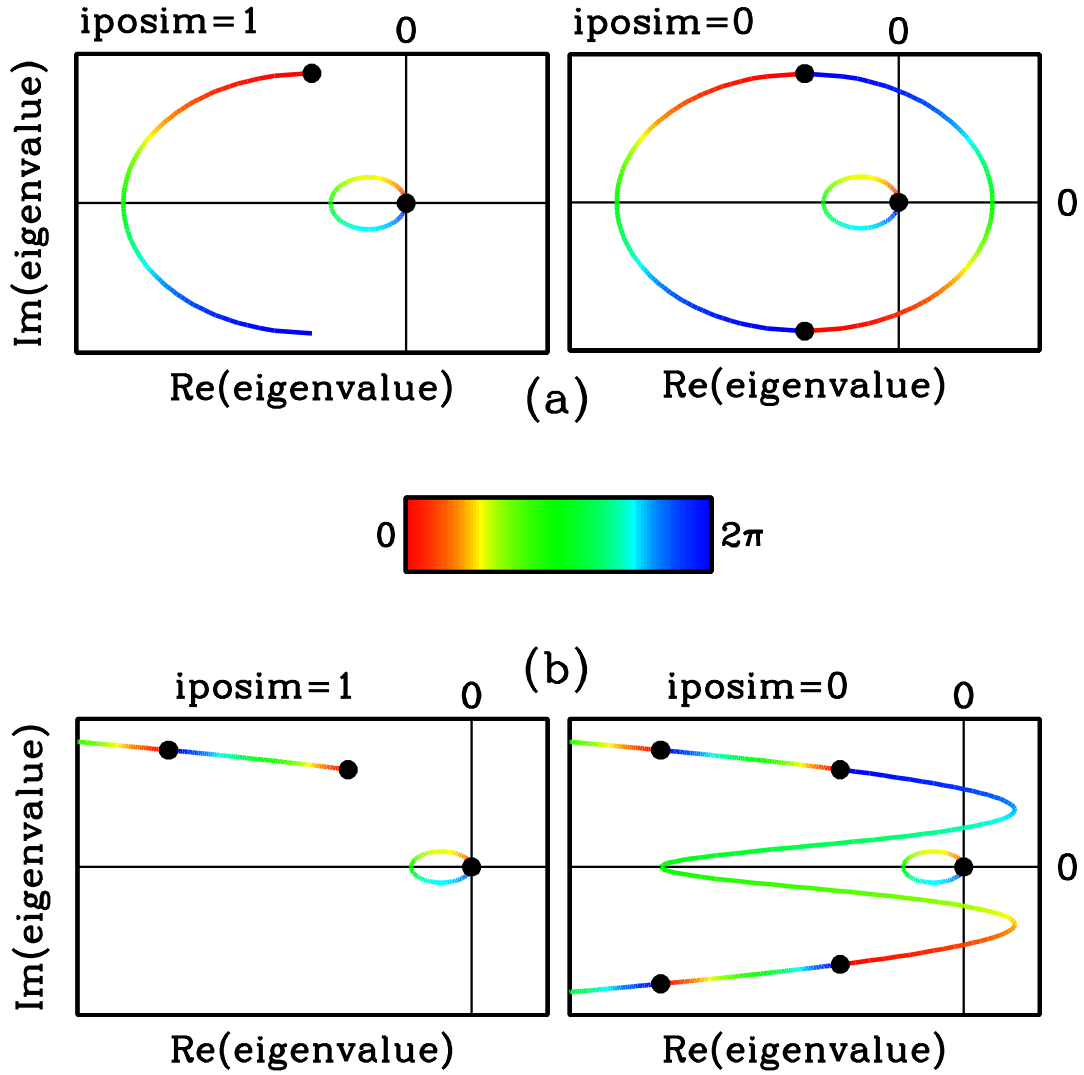


Figure 3.4: A schematic illustration of two spectra for which the setting $\text{iposim}=1$ leads to an incorrect conclusion about periodic travelling wave stability. The black dots indicate eigenvalues corresponding to periodic eigenfunctions. In both (a) and (b) the part of the spectrum illustrated consists of two branches, one of which crosses the imaginary axis, implying that the periodic travelling wave is unstable. However for $\text{iposim}=1$ only the eigenvalues corresponding to periodic eigenfunctions and with imaginary part ≥ 0 are used as starting points for continuation of the spectrum. Consequently in both (a) and (b) the part of the spectrum that crosses into the right hand half plane is not calculated when $\text{iposim}=1$. The colours along the spectra indicate the phase difference in the eigenfunction over one period of the periodic travelling wave.

shown in part (b) of the figure, the part of the spectrum crossing into the right hand half plane is not calculated when `iposim=1`.

If `iposim=1`, WAVETRAIN runs two tests in an attempt to determine whether part of the spectrum in the upper half plane has been omitted from the calculations. These tests are not fool-proof, and a brief summary of them may be helpful to users who are wondering why WAVETRAIN suggests that they change `iposim` from 1 to 0. For each calculated segment of the spectrum, WAVETRAIN records the start and end points, which correspond to $\gamma = 0$ and $\gamma = 2\pi$ respectively. Here, as previously, γ is the phase difference in the eigenfunction across one period of the periodic travelling wave: this is the principle continuation parameter along the spectrum. Note that the start point will be close to a matrix eigenvalue corresponding to a periodic eigenfunction, but the match will not be exact because of errors introduced by the discretisation used to form the matrix (see §3.1.13).

If the imaginary part of the eigenvalue at $\gamma = 2\pi$ is negative, then the subsequent part of the spectrum (in the sense of increasing γ) will not be calculated when `iposim=1`, and will cross into the upper half plane. This case is illustrated in Figure 3.4a, and WAVETRAIN outputs an error message in such a case. To avoid problems arising from numerical accuracy, the actual criterion used by WAVETRAIN for an error message is that the imaginary part of the eigenvalue at $\gamma = 2\pi$ is less than -10evzero . Here the factor of 10 is included to reduce the likelihood of inappropriate error messages. The constant `evzero` is set in `defaults.input`.

To describe the second test performed by WAVETRAIN, consider a calculated portion \mathcal{S} of the spectrum, with start point $\mathcal{S}_{\gamma=0}$ and end point $\mathcal{S}_{\gamma=2\pi}$. WAVETRAIN determines whether $\mathcal{S}_{\gamma=0}$ corresponds to the end point of another calculated portion of the spectrum. There are four possible reasons why this would not be the case.

- (i) $\text{Im } \mathcal{S}_{\gamma=0} = 0$. Then $\mathcal{S}_{\gamma=0} = 0$ might be the end point of a part of the spectrum in the lower half plane, which is not calculated when `iposim=1`.
- (ii) $\mathcal{S}_{\gamma=0}$ is the end point of a part of the spectrum that was not calculated by WAVETRAIN because of a convergence error. Most often this arises because the matrix eigenvalues are a poor approximation to the actual eigenvalues corresponding to periodic eigenfunctions (see §3.1.13).
- (iii) WAVETRAIN only calculates the part of the spectrum corresponding to the `nevals` eigenvalues with largest real part and (for `iposim=1`) zero or positive imaginary part. $\mathcal{S}_{\gamma=0}$ might be the end point of a portion of the spectrum that has not been calculated for this reason.
- (iv) $\mathcal{S}_{\gamma=0} = 0$ might be the end point of a portion of the spectrum that originates in the lower half plane. This case is illustrated in parts (a) and (b) of Figure 3.4.

It is only in case (iv) that a part of the spectrum in the upper half plane is not calculated because of the setting `iposim=0`. Unfortunately it is not possible to systematically distinguish between cases (iii) and (iv). Therefore WAVETRAIN attempts an approximate distinction: it generates an error message about the value of `iposim` if $\mathcal{S}_{\gamma=0}$ is not the end point of any of the calculated parts of the spectrum, and also if $\text{Im } \mathcal{S}_{\gamma=0} > 0$, if there

are no convergence errors, and if $\text{Re } \mathcal{S}_{\gamma=0} > \text{Re } \mathcal{S}_{\gamma=2\pi}$. To avoid problems arising from numerical accuracy, WAVETRAIN designates the start point of one part of the spectrum and the end point of another to be the same if their difference has modulus less than `evzero`. Also in an attempt to avoid numerical artifacts, WAVETRAIN uses the inequality $\text{Im } \mathcal{S}_{\gamma=0} > 10 \text{ evzero}$ rather than $\text{Im } \mathcal{S}_{\gamma=0} > 0$; the factor of 10 is included to reduce the likelihood of inappropriate error messages.

These tests detect most cases in which the setting `iposim` = 1 might cause an incorrect conclusion about wave stability, but they are not fool-proof. They may lead to error messages when there is not a missing part of the spectrum, and they may also fail to detect a case in which there is a missing part. Therefore it is advisable to visually inspect spectra when using the setting `iposim` = 1, at least for some representative parameter sets.

If the tests detect a possible problem then WAVETRAIN will display a “possible error” message. The user then has two options. One is to investigate, by plotting the spectrum. The other option is to simply rerun with `iposim`=0, which will definitely avoid the problem if there is one. In order to calculate the same part of the spectrum it will be necessary to increase `nevals` to $2\text{nevals}^{\text{old}} - n_0$, where n_0 is the number of real eigenvalues corresponding to periodic eigenfunctions. Note that $n_0 \geq 1$ since zero is always an eigenvalue, corresponding to the neutral stability of the periodic travelling wave to translation. Therefore it is often easiest to simply increase `nevals` to $2\text{nevals}^{\text{old}} - 1$. The downside of this change is that the run time will approximately double.

3.1.9 The Parameter `iwave`

A major potential complication for calculations using WAVETRAIN is that there may be folds in the branch of periodic travelling wave solutions, implying that there is more than one wave solution for some pairs of control parameter and wave speed values. An example of this is provided by the files in the `demo1` input subdirectory. This problem has the same equations and the same values of the “other parameters” as for `demo`, but considers different ranges for the control parameter A and the wave speed c , and uses different settings in `constants.input`. Thus the input files in `demo` and `demo1` are identical except for `parameter_range.input` and `constants.input`. Looping through the control parameter–wave speed plane in the usual way using the `ptw_loop` command shows that periodic travelling wave solutions exist in a thin diagonal strip in this parameter plane. The results of the `ptw_loop` command for the `parameter_range.input` file provided at installation are illustrated in Figure 3.5a. Note that during this run and several of the subsequent runs for the problem `demo1`, a warning message about a small number of continuation steps will appear. This can be ignored: it is a consequence of the rather large step sizes that are set for the illustrative problems `demo` and `demo1`. Hopf bifurcation and homoclinic solution loci can be calculated in the usual way, via the commands

```
hopf_locus demo1 101 1.065 21.0 1.08 21.0
```

and

```
period_contour demo1 101 period=3000 1.085 20.0 1.08 20.0
```

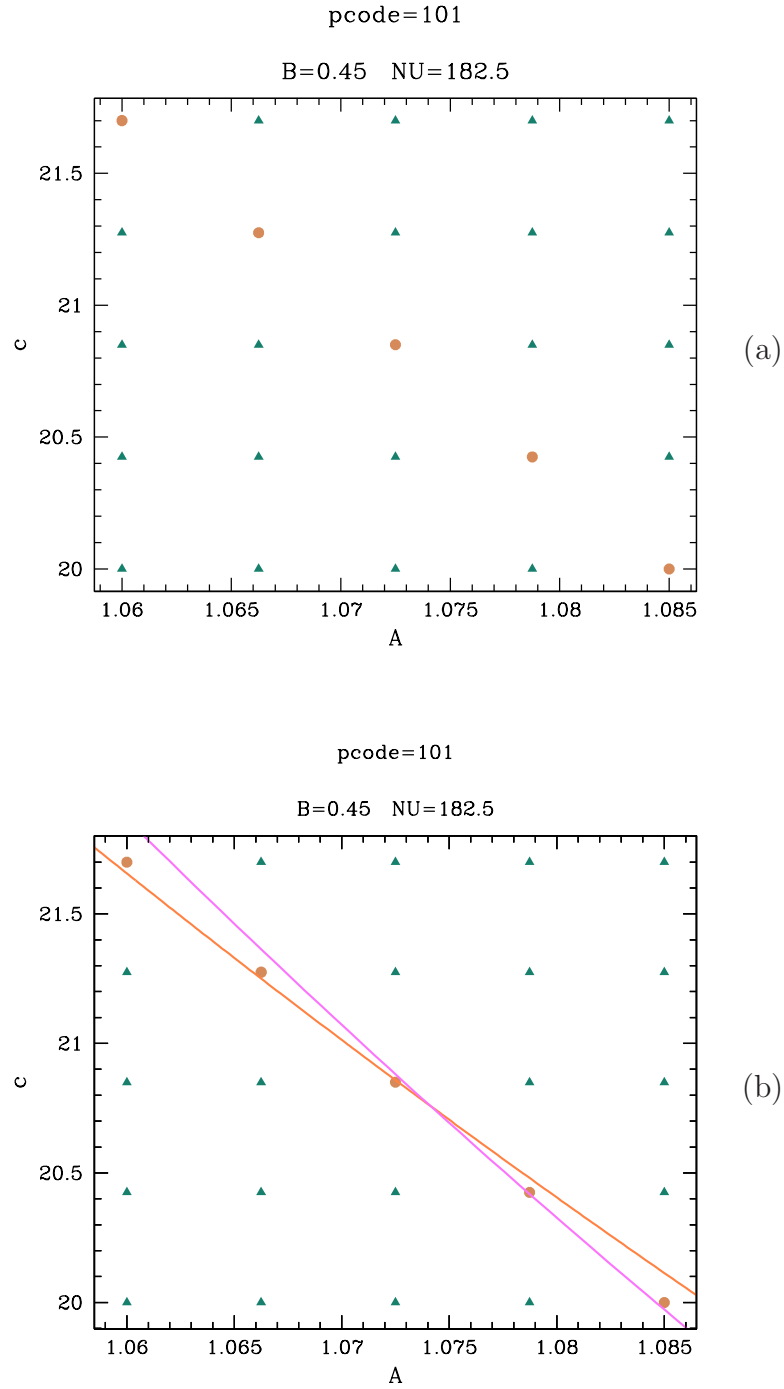


Figure 3.5: (a) Results on the existence of periodic travelling wave solutions for the problem **demo1**. (b) The same plot as in (a) but with the addition of the Hopf bifurcation locus and (an approximation to) the locus of homoclinic solutions. A key illustrating the meaning of the various colours and symbols is shown in Figure 2.4 on page 18. The run and plot commands used to generate these figures are given in the main text, and are also listed in the Appendix.

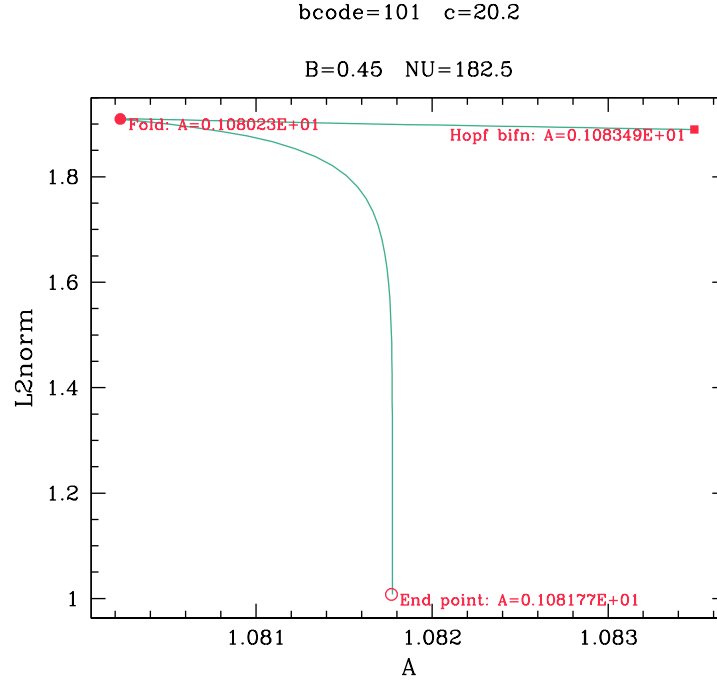


Figure 3.6: A typical bifurcation diagram for the problem `demo1`. The run and plot commands used to generate this figure are given in the main text, and are also listed in the Appendix.

followed by

```
set_homoclinic demo1 101 101
```

The results are illustrated in Figure 3.5b, showing that these loci intersect one another. This indicates a somewhat complicated solution structure, which is best investigated using the `bifurcation_diagram` command. Figure 3.6 illustrates a typical result from such a study, generated by the command

```
bifurcation_diagram demo1 c=20.2
```

which shows that there is a fold in the periodic travelling wave solution branch, with two different periodic travelling wave solutions for some values of A .

When there are such multiple solutions, WAVETRAIN is able to study all of them via the parameter `iwave`. In all previous examples `iwave` has been set to 1, which causes WAVETRAIN to study the first wave solution along the solution branch for a given pair of A and c values. Here the counting of wave solutions is done as the control parameter varies, starting from the solution located by the wave search method specified in `equations.input`. Setting `iwave=2` causes WAVETRAIN to study instead the second wave solution for the given parameters.

To illustrate this, Figure 3.7a shows the results of a run of `ptw_loop` for `demo1` for a finer grid of control parameter and wave speed values than used in Figure 3.5a, but again with `iwave=1`. To perform this run, `parameter_range.input` has been edited, with the grid reset to 50×15 , but all other constants have been left as they were at installation. Note that these grid dimensions mean that `ptw_loop` attempts to calculate periodic travelling wave solutions for 750 pairs of control parameter and wave speed values, and consequently the run time for this command is relatively long (about an hour on a typical desktop computer). In addition, the command `plot demo1 102` takes some time to perform its initial setups (about 5 minutes on a typical desktop computer) because of the large volume of data that is being processed. Figure 3.7a shows that there is a strip of the parameter plane in which there are periodic travelling wave solutions; the left hand boundary of this strip is composed partly of the Hopf bifurcation locus, partly of the homoclinic solution locus, and partly of the fold in the bifurcation diagram. The command `fold_locus` for tracing the locus of such a fold is described in §3.1.10.

If one edits the file `constants.input` and changes `iwave` to 2, and then reruns `ptw_loop`, one finds a thinner strip of the parameter plane in which there are periodic travelling waves (illustrated in Figure 3.7b). This strip is bounded by the locus of folds (on the left) and the homoclinic locus (on the right). This thin strip is the region of the parameter plane in which there are two different periodic travelling waves. If one were to proceed to calculate wave stability with `iwave` still set to 2, it is the stability of this second wave that would be calculated. Similarly, calculations done using the `period_contour` and `stability_boundary` commands would refer to the second wave along the solution branch. Setting `iwave > 2` for `demo1` would simply show that there is no part of the parameter plane in which there are more than two periodic travelling waves.

Before leaving the use of WAVETRAIN with `iwave > 1`, it is important to mention a significant complication. When numerical continuation along a solution branch reaches an end point, it sometimes reverses and continues back along the same branch. This is particularly common when a branch of periodic travelling wave solutions connects two different Hopf bifurcation points: the continuation will often run repeatedly backwards and forwards between the two points. Consequently, the solution branch will contain a sequence of periodic travelling waves for any given pair of control parameter–wave speed values along it, although each of these waves will be the same. It is necessary for WAVETRAIN to distinguish such “false positive” occurrences of multiple wave solutions from genuine cases such as those described above for `demo1`. This is achieved using the parameter `wavefrac`, which is set in `constants.input`, although it is an optional entry, with `wavefrac = 0.001` being used if it is omitted. When WAVETRAIN detects a second or subsequent wave along the solution branch for the specified parameter value, it compares the L2-norm and period with those of the previous wave solution. If the difference divided by the mean is less than `wavefrac` for both the L2-norm and the period, then WAVETRAIN deems the two solutions to be the same and terminates the continuation.

The parameter `iwave` can also be set to zero. To illustrate the meaning of this special setting, consider the example problem `demo` discussed in §2.1. The selected region of the control parameter–wave speed plane has a significant region to the right of the Hopf bifurcation locus in which there are no periodic travelling wave solutions. If `iwave=1`, searching for a wave for control parameter and wave speed values in this region would

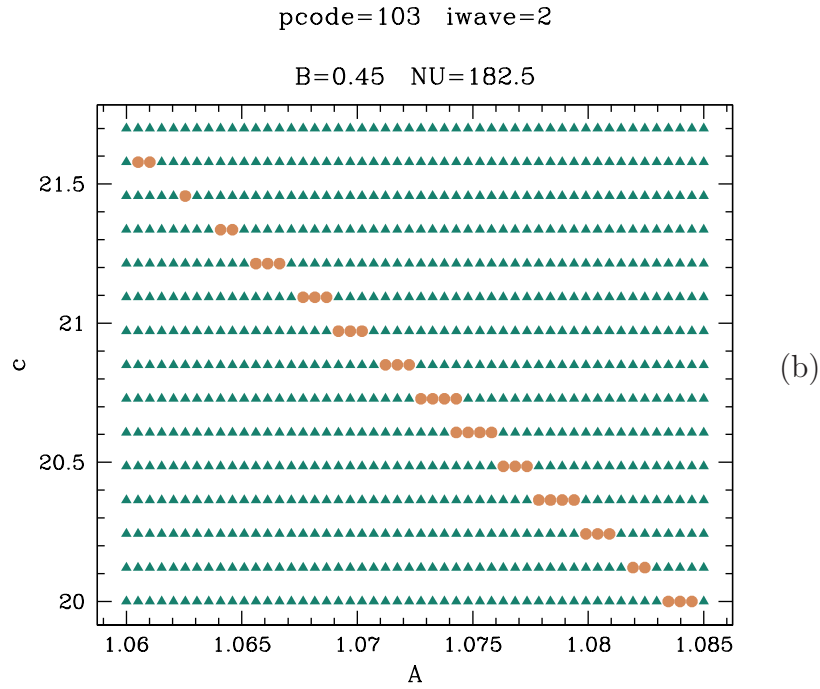
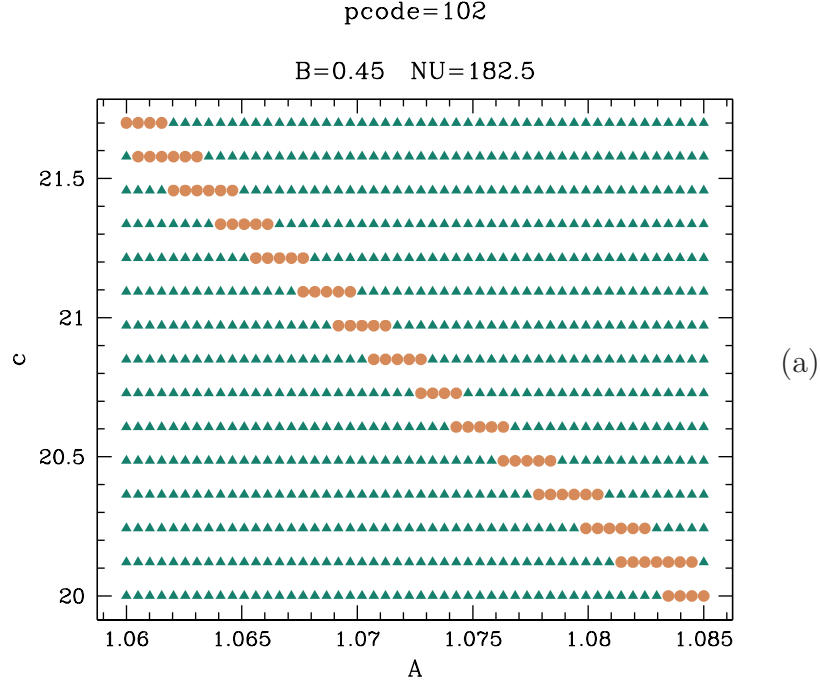


Figure 3.7: Results on the existence of periodic travelling wave solutions for the problem `demo1`, using a fine grid of control parameter–wave speed values. (a) `iwave=1`; (b) `iwave=2`. A key illustrating the meaning of the various colours and symbols is shown in Figure 2.4 on page 18. The run and plot commands used to generate these figures are given in the main text, and are also listed in the Appendix.

proceed as follows. First, WAVETRAIN would vary the control parameter across the search region for Hopf bifurcations (specified in `equations.input`). If a Hopf bifurcation is detected, WAVETRAIN would then vary the control parameter along the periodic travelling wave solution branch, searching for the required value of the control parameter. However, since there is no periodic travelling wave solution, this search only ends when the numerical continuation ends because of a convergence failure close to the homoclinic solution locus. If one knew, or assumed, that the Hopf bifurcation was supercritical then one could eliminate this entire computation, since one would know *a priori* that there are no periodic travelling wave solutions to the right of the Hopf bifurcation locus. This assumption can be implemented by setting `iwave=0`. The file `equations.input` contains the start and end values of the range of control parameter values over which to search for a Hopf bifurcation in the travelling wave equations (see §2.2.2). The specific consequence of setting `iwave=0` is that the end value is not used: it is replaced by the value of the control parameter at which a periodic travelling wave is sought. For the problem `demo`, if one edits `equations.input` changing `iwave` from 1 to 0 and then reruns the `ptw_loop` command (followed by calculation of Hopf bifurcation loci and period contours as in §2.1), the result is as shown in Figure 3.8; this should be compared with Figure 3.1 on page 77. The overall structure of the parameter plane is the same, but different outcome symbols are now used on the right and left hand sides of the region of periodic travelling waves. The blue-green triangles on the left hand side indicate that the search for a periodic travelling wave ended with a convergence failure in the numerical continuation. Typically this corresponds to the branch of periodic travelling wave solutions approaching a homoclinic solution, though of course it might indicate a real error. The new symbol (a purple square) indicates that the search for a periodic travelling wave ended without a convergence failure. This occurs most often when `iwave` has been set to 0 and no Hopf bifurcation is found, and this is the case for Figure 3.8. However, it could occur for other reasons, for example if the values of `ds`, `dsmin` and `dsmax` are too small, so that continuation ends prematurely (but without a convergence error). In all cases, more detailed information on the outcome of the run is available in the file `outcome.data` in the relevant output subdirectory (see §4.4); this indicates which of the different possibilities has occurred. Note that if the user wishes to include a key in material for presentation or publication, then they will probably prefer to use a single entry for cases where there is no periodic travelling wave, and correspondingly to have a single entry in the key. This can be achieved by changing the plotter setting `pplanekeytype` (see §3.3.1).

The key advantage of setting `iwave` to 0 is that the run time for the loop is lower. However, care must be exercised when using this setting, because it would give misleading results if the Hopf bifurcation was subcritical or had a fold.

Further details of the problem studied using `demo1` are given in Sherratt (2011).

3.1.10 Tracing the Loci of Folds

When there is a fold in the branch of periodic travelling wave solutions, the locus of this fold often forms part of the boundary of region of the control parameter–wave speed plane in which periodic travelling waves exist. WAVETRAIN can calculate such a locus

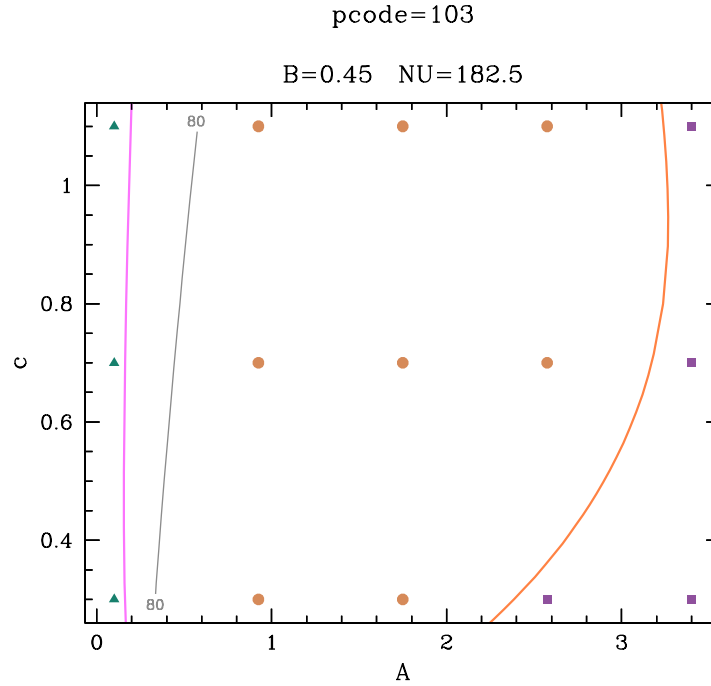


Figure 3.8: An illustration of the effect of setting the constant `iwave` to 0. This figure should be compared with Figure 3.1 on page 77, which shows the corresponding results for `iwave`=1. A key illustrating the meaning of the various colours and symbols is shown in Figure 2.4 on page 18. The run and plot commands used to generate this figure are given in the main text, and are also listed in the Appendix.

using the command `fold_locus`. This is illustrated by returning to the `demo1` example discussed in §3.1.9. Recall that periodic travelling waves exist in a thin strip in the A - c plane (illustrated in Figure 3.7a); this strip is bounded by segment of the locus of Hopf bifurcation points, the locus of homoclinic solutions, and the locus of folds. To calculate the last of these, one enters

```
fold_locus demo1 102 1.07 21 1.05 21
```

where either the two control parameter values (third and fifth arguments) or the two wave speeds (fourth and sixth arguments) must be the same, as for `hopf_locus` and `stability_boundary`. In executing this command, `WAVETRAIN` first calculates the periodic travelling wave at $A = 1.07$ and $c = 21$. It then performs a numerical continuation along the periodic travelling wave solution branch with $c = 21$ fixed, and with A changing in the direction of the fifth argument 1.05: so A is decreased. `WAVETRAIN` monitors the solution branch for a fold, and once this has been located, its locus is tracked in the A - c plane. The run is given a 3 digit “fcode” value (101-999).

When plotting the fold locus, it is convenient to plot also the Hopf bifurcation and homoclinic solution loci, but these were calculated for a different `pcode` value. Therefore one must copy them across, via the commands

```
copy_hopf_loci demo 101 102
```

and

```
copy_period_contours demo 101 102
```

after which the WAVETRAIN plotter can be started by typing `plot demo1 102`. One recreates Figure 3.7 via the command `@pplane`. Hopf bifurcation and homoclinic loci are added to the plot by typing

```
@hopf_locus 101
```

and

```
@period_contour 101
```

and finally the fold locus is added by the command

```
@fold_locus 101
```

where 101 is the `fcode` value. The resulting plot is shown in Figure 3.9.

Users should be aware of the following points when using the `fold_locus` command.

- Since the initial step in the calculation of a fold locus is a calculation of the periodic travelling wave at the control parameter and wave speed values given in the third and fourth arguments, these arguments must be chosen so that there is a periodic travelling wave.
- The value of `iwave` (see §3.1.9) is significant for `fold_locus`, since it affects the periodic travelling wave for the third and fourth arguments, which is the initial step in the calculation. For example if `iwave` is set to 3, then `fold_locus` will first calculate the third periodic travelling wave along the solution branch with control parameter and wave speed given by the third and fourth arguments. It then locates the fold adjacent to this point on the solution branch, in the direction determined by the fifth and sixth arguments. It is the locus of this fold that is calculated by `fold_locus`.
- WAVETRAIN has a command `copy_fold_loci` to copy fold loci from one `pcode` to another. It is directly analogous to the commands `copy_stability_boundaries`, `copy_hopf_loci` and `copy_period_contours`.

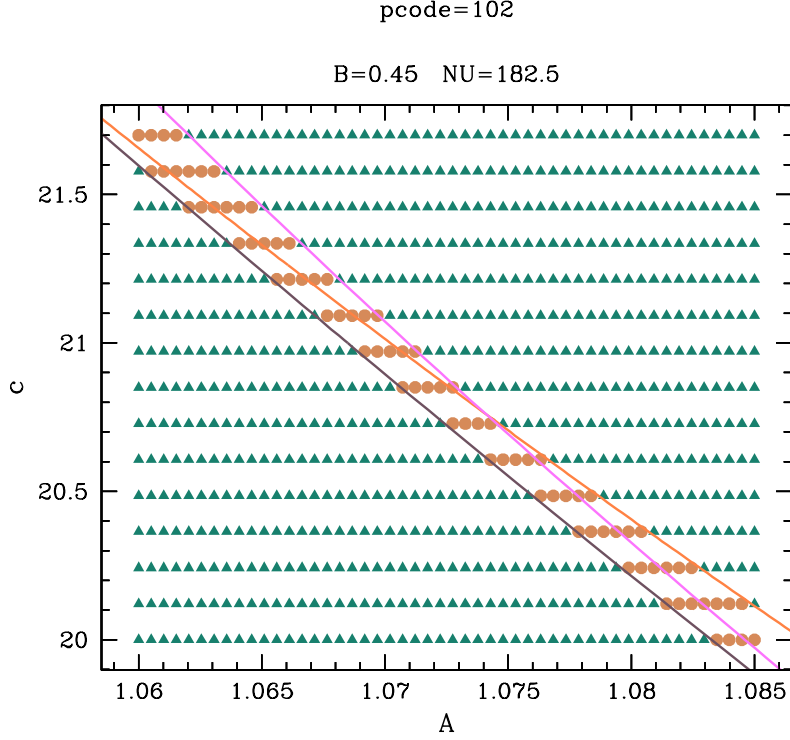


Figure 3.9: Results on the existence of periodic travelling wave solutions for the problem `demo1`. The plot is the same as that in Figure 3.7a but with the addition of the Hopf bifurcation locus, (an approximation to) the locus of homoclinic solutions, and the loci of folds in the periodic travelling wave solution branch. A key illustrating the meaning of the various colours and symbols is shown in Figure 2.4 on page 18. The run and plot commands used to generate these figures are given in the main text, and are also listed in the Appendix.

3.1.11 Commands for Deleting Data Files

WAVETRAIN generates a large number of data files, in a large number of output subdirectories. Manual deletion of unwanted files and directories would be very laborious, and thus WAVETRAIN provides a series of commands for systematic deletion. The command

```
rmrcode demo 1002
```

deletes all of the data files associated with the run of `ptw` or `stability` that was allocated the rcode value 1002. For runs associated with loops through the control parameter–wave speed plane (via the `ptw_loop` or `stability_loop` commands), the pcode value must also be specified; thus

```
rmrcode demo 101 1002
```

deletes the run with rcode 1002 for pcode 101. Note that in all WAVETRAIN commands, if a pcode value is needed as an argument, it is always given as the second argument, following the subdirectory name. One final point concerning the command `rmrcode` is that for the purposes of file classification (see §4.4.1), all runs of `ptw` and `spectrum` are allocated to the dummy pcode value of 100. Thus the commands

```
rmrcode demo 102
```

and

```
rmrcode demo 100 102
```

are equivalent.

Sometimes one may wish to delete files associated with all rcode values, for a particular pcode. To achieve this, one replaces the rcode number by `all` in the above commands. (`All` or `ALL` are also allowed).

The commands `rmccode`, `rmhcode`, `rmfcode` and `rmscode` are directly analogous except that a pcode value (≥ 101) is required, since period contours, Hopf bifurcation loci, fold loci and stability boundaries are necessarily associated with parameter plane runs. Thus

```
rmhcode demo 107 102
```

deletes all files associated with the Hopf bifurcation locus with hcode 102 for pcode 107, while

```
rmscode 112 all
```

deletes all files associated with all stability boundaries, for pcode 112. The commands `rmrcode` and `rmecode` work in a similar way, but in this case a pcode value is never given since bifurcation diagram and eigenvalue convergence calculations are not associated with control parameter–wave speed planes. Thus

```
rmrcode demo 108
```

deletes all files associated with the bifurcation diagram with bcode 108, while

```
rmecode demo all
```

deletes all files associated with all eigenvalue convergence calculations.

To delete all files associated with a particular pcode, one can use the command `rmrcode`, for example

```
rmrcode demo 103
```

or

```
rmrcode demo 100
```

(the second of these will delete all files associated with any runs of `ptw`, `stability`, `bifurcation_diagram` and `eigenvalue_convergence`). The usage

```
rmrcode demo all
```

is also allowed; this will delete all output associated with the input subdirectory **demo**, and confirmation is requested to prevent unwanted deletion of a large amount of output.

A related command is **rmeps**, which is used to delete files in the **postscript_files** directory. For example, the command

```
rmeps demo demoplot
```

will delete the file **demoplot.eps** in the **demo** subdirectory of **postscript_files**, plus the associated plotting record if there is one. (Plotting records are described in §3.3.6). The usage

```
rmeps demo all
```

is also allowed. It is possible that a file named **all.eps** has been created in the **demo** subdirectory of **postscript_files**. If so, this will be deleted, together with any associated plotting record. If there is no such file, then all of the files in this postscript files subdirectory will be deleted, plus any associated plotting records.

3.1.12 Setting the Wave Search Method and Hopf Bifurcation Search Range in Complicated Cases

As well as listing various equations, the file **equations.input** specifies the method used by WAVETRAIN to locate a periodic travelling wave for given control parameter and wave speed values. In many cases (such as for the problem **demo** discussed in §2.1) there is exactly one Hopf bifurcation in the travelling wave equations for any given value of the wave speed. Then a “direct” method can be used, meaning that WAVETRAIN searches for a Hopf bifurcation at the wave speed required for the periodic travelling wave, and then proceeds “directly”, via a single numerical continuation, to the required wave solution. The control parameter range used in the Hopf bifurcation search is specified in the two lines following the word “direct” in **equations.input**. For **demo** a suitable search range has a starting value of $2.001B$, since the steady state undergoing the Hopf bifurcation only exists for $A \geq 2B^\dagger$. Here A is the name assigned to the control parameter in **variables.input**, and B is one of the parameters specified in **other_parameters.input**. The end value of the search range is 3.8, which is the upper limit on A in **parameter_range.input**; note however that the Hopf bifurcation search range is not restricted to the range specified in **parameter_range.input**.

The real importance of the Hopf bifurcation search range is for more complex situations in which there is more than one Hopf bifurcation as the control parameter is varied with a fixed wave speed. In the absence of an example problem that provides an adequate illustration of the various possibilities, this discussion is illustrated using a sketch (Figure 3.10). Suppose that for a particular problem, periodic travelling waves exist within

[†]The actual condition for steady state existence is $A \geq 2B$. However, there is a saddle-node bifurcation at $A = 2B$, with two steady states for $A > 2B$, only one of which is of interest in the problem **demo**. To avoid WAVETRAIN following the wrong steady state branch when searching for a Hopf bifurcation, the problem **demo** has been formulated following the advice in the comments in **equations.input**, with the condition for steady state existence being set slightly away from the bifurcation point, specifically $A \geq 2.001B$. Correspondingly, the end point for the Hopf bifurcation search range is set to $A = 2.001B$ rather than $A = 2B$.

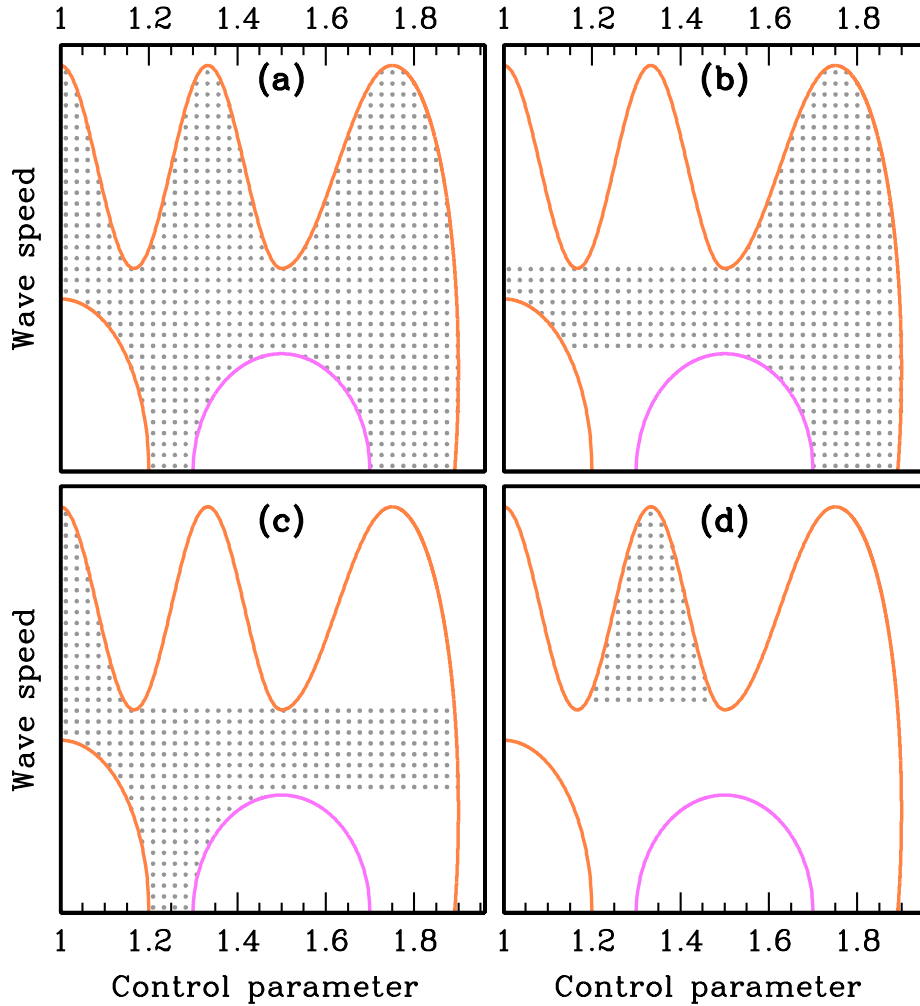


Figure 3.10: An illustration of the effects of changing the Hopf bifurcation search range, which is set in `equations.input`. This figure is a sketch. (a) The spotted region is the parameter region in which there are periodic travelling waves. For simplicity it is assumed that there is exactly one periodic travelling wave solution throughout this region, and that WAVETRAIN is being run with `iwave=1` (see §3.1.9). (b–d) The spotted region is the parameter region in which periodic travelling waves are accessible to WAVETRAIN for three different Hopf bifurcation search ranges, using a “direct” wave search method in all three cases: (b) 1.95–1.0; (c) 1.0–1.95; (d) 1.25–1.5 or 1.5–1.25. The orange curves represent loci of Hopf bifurcations in the travelling wave equations, and the pink curves indicate loci of homoclinic solutions, as in the key in Figure 2.4.

the region of the control parameter–wave speed plane that is spotted in Figure 3.10a. For simplicity, assume that there is exactly one periodic travelling wave solution throughout this region, and that WAVETRAIN is being run with `iwave=1` (see §3.1.9 for information on `iwave`). One might at first specify a direct wave search method with the Hopf bifurcation search range having a start value of 2.0 and an end value of 1.0. Running `ptw` or `spectrum` would then cause WAVETRAIN to stop at the first Hopf bifurcation it encountered as the control parameter decreased from 2.0 to 1.0, and to then continue travelling wave solutions from that point. This would lead to the detection of periodic travelling waves in the parameter region that is spotted in Figure 3.10b. Note that the order of the Hopf bifurcation search end points in `equations.input` is important. If instead one specified a search range with a start value of 1.0 and an end value of 2.0, the parameter region in which periodic travelling waves were detected would be the spotted region in Figure 3.10c.

Plotting the results from a study with either of these Hopf bifurcation search ranges would immediately suggest that an investigation using the `bifurcation_diagram` command would be helpful. Crucially, this command does not use the wave search method and Hopf bifurcation search range specified in `equations.input`. Rather, it scans the parameter range specified in `parameter_range.input` and continues the branch of periodic travelling wave solutions emanating from each Hopf bifurcation that is detected. Running `bifurcation_diagram` for a series of different values of the wave speed would reveal the true structure of the parameter plane, which could be clarified further by calculating the loci of Hopf bifurcation points and homoclinic solutions. This would show that there are regions of the control parameter–wave speed plane in which periodic travelling waves exist, but which are inaccessible to WAVETRAIN with a direct search method and a Hopf bifurcation search range with start and end values of 2.0 and 1.0. To calculate wave forms and stability in these regions, one would use a different Hopf bifurcation search range. For example, search range start and end values of 1.25 and 1.5 (in either order, and again with the direct wave search method) would enable WAVETRAIN to access periodic travelling wave solutions in the part of the control parameter–wave speed plane that is spotted in Figure 3.10d.

The term “direct” is used for the wave search methods discussed above because, once the Hopf bifurcation point has been found, WAVETRAIN locates the periodic travelling wave in a single numerical continuation, simply varying the control parameter along the periodic travelling wave solution branch until the required value is reached. However, there are some situations in which this simple approach cannot be used, and an example is illustrated in the sketch in Figure 3.11. Again periodic travelling waves exist within the region of the control parameter–wave speed plane that is spotted. For values of the wave speed less than 1, the periodic travelling waves can easily be located using a “direct” search method, as discussed above. However for wave speeds greater than 1, there is no Hopf bifurcation for any value of the control parameter, and thus this method cannot be used. WAVETRAIN offers the “indirect” search method for this situation. Here the Hopf bifurcation search occurs by varying the wave speed within a specified range, at a specified value of the control parameter. For instance, suppose that one wished to study a periodic travelling wave for control parameter 1.8 and wave speed 1.2, illustrated by the large red dot in Figure 3.11. The user might specify a Hopf bifurcation search between

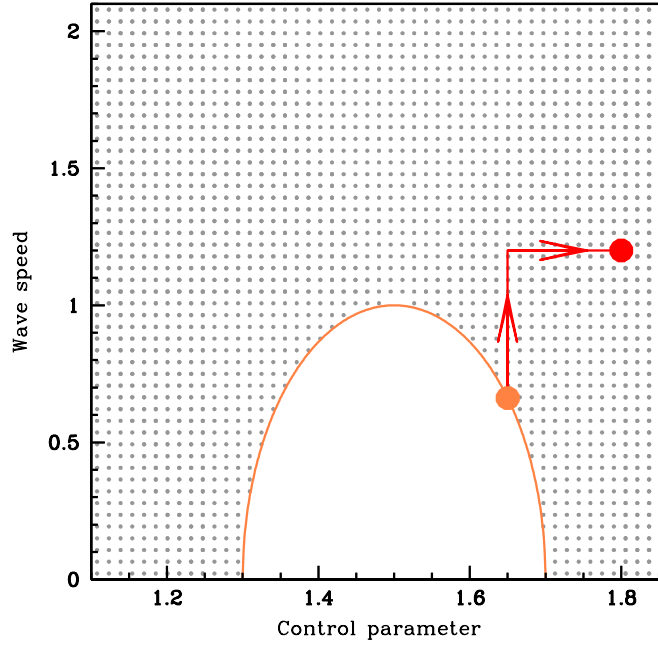


Figure 3.11: An illustration of an example requiring an “indirect” wave search method. This figure is a sketch. The spotted region is the parameter region in which there are periodic travelling waves. The orange curve represents a locus of Hopf bifurcations in the travelling wave equations, as in the key in Figure 2.4. The red lines indicate the wave search method, which starts from a Hopf bifurcation at the point indicated by the large orange dot, and proceeds first with a constant control parameter value and then with a constant wave speed, reaching the required values (indicated by the large red dot) “indirectly”.

wave speeds 0.5 and 0.8, at control parameter value 1.65. WAVETRAIN would then find the Hopf bifurcation point indicated by the large orange dot in Figure 3.11. It would then perform two separate numerical continuations to locate the required wave. Firstly the periodic travelling wave solution branch would be followed, varying the wave speed with the control parameter fixed, until the required value is reached. Then the control parameter would be varied, with fixed wave speed, until the required pair of values is reached.

It is possible that there is not a Hopf bifurcation for any pair of control parameter and wave speed values in the parameter plane being considered; then neither the direct nor indirect wave search methods will work. To accommodate this situation, WAVETRAIN provides a third wave search method, termed the “file” method. In this case, the user must supply a periodic travelling wave solution for one pair of control parameter and

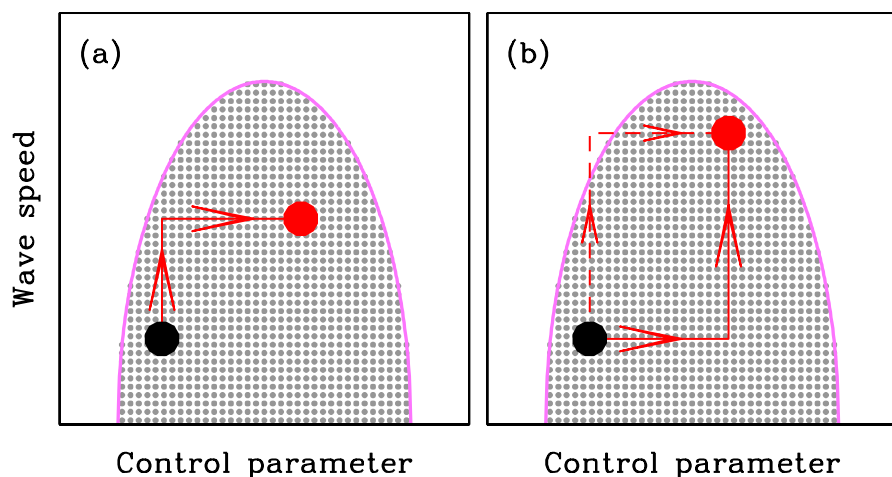


Figure 3.12: An illustration of the “file” wave search method. This figure is a sketch. The spotted region is the parameter region in which there are periodic travelling waves. The pink curve represents a locus of homoclinic solutions of the travelling wave equations, as in the key in Figure 2.4. The black dot indicates the control parameter and wave speed values for which a periodic travelling wave solution has been provided by the user, in the file `ptwsoln.input`. The red dot indicates the control parameter and wave speed values for which a periodic travelling wave is required. In (a), WAVETRAIN first varies the wave speed until the required value is reached, with control parameter fixed, and then varies the control parameter with fixed wave speed, to reach the required pair of values (indicated by the red lines and arrows). However in (b) this approach fails because the homoclinic locus is reached before the required value of the wave speed (indicated by the dashed red lines and arrows). WAVETRAIN then attempts the steps in the opposite order, first varying the control parameter with fixed wave speed, and then varying the wave speed with fixed control parameter; this second approach is successful (indicated by the solid red lines and arrows).

wave speed values; typically this would come from numerical simulations of the partial differential equations. The control parameter and wave speed values for this solution are given in `equations.input`, and the solution itself is specified in the additional input file `ptwsoln.input`, which must be placed in the relevant input subdirectory. The required format for `ptwsoln.input` is described in §2.2.2. Using this solution as a starting point, WAVETRAIN first follows the periodic travelling wave solution branch with the control parameter fixed, varying the wave speed until the required value is reached. Then the control parameter is varied, with fixed wave speed, until the required pair of values is reached (Figure 3.12a). If this procedure fails, then WAVETRAIN tries the steps in the opposite order, first varying the control parameter with fixed wave speed, and then varying the wave speed with a fixed value of the control parameter (Figure 3.12b).

Two points should be noted in connection with the “file” wave search method. Firstly, if `iwave > 1`, it is always the control parameter that is varied to look for the second

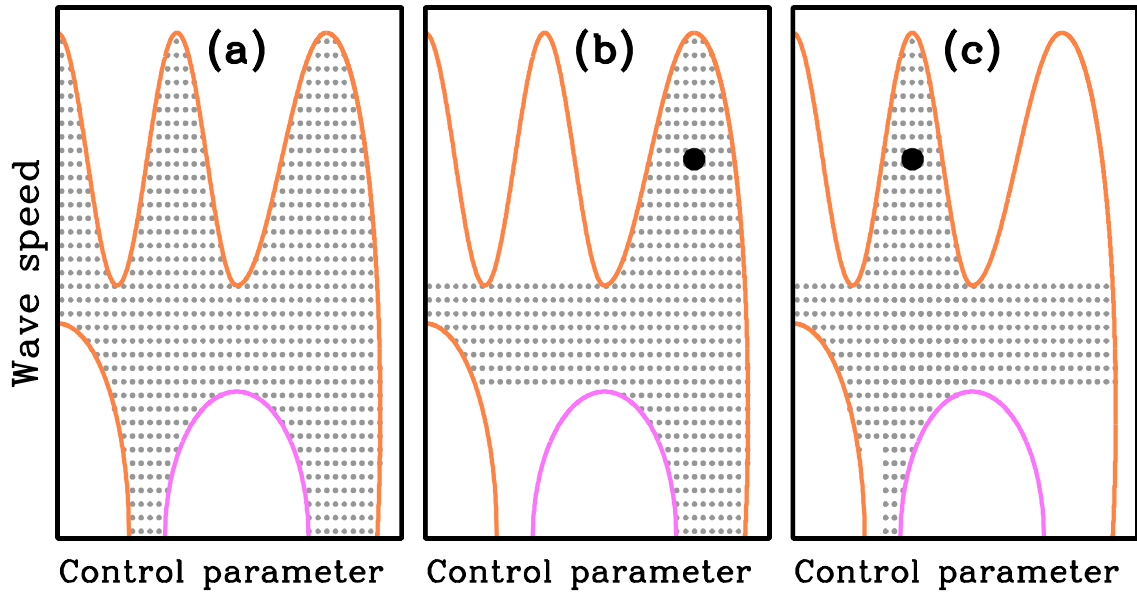


Figure 3.13: An illustration of the effects of different starting points for the “file” wave search method. This figure is a sketch. (a) The spotted region is the parameter region in which there are periodic travelling waves. For simplicity it is assumed that there is exactly one periodic travelling wave solution throughout this region, and that WAVETRAIN is being run with `iwave=1` (see §3.1.9). (b,c) The spotted regions are those in which periodic travelling waves are accessible to WAVETRAIN for two different starting points, using the “file” wave search method. The orange curves represent loci of Hopf bifurcations in the travelling wave equations, and the pink curves indicate loci of homoclinic solutions, as in the key in Figure 2.4.

and subsequent waves at the specified pair of control parameter and wave speed values, irrespective of the order in which the control parameter and wave speed have been varied to locate the first wave. Secondly, if the boundary of the parameter region giving periodic travelling waves is complicated then different parts of that region may be accessed by different starting points. This is illustrated by the sketch in Figure 3.13. The spotted region in Figure 3.13a is that in which periodic travelling waves exist; as for Figure 3.10, it is assumed that there is exactly one periodic travelling wave solution throughout this region, and that WAVETRAIN is being run with `iwave=1` (see §3.1.9 for information on `iwave`). In figures 3.13b and 3.13c, the spotted regions are those in which the periodic travelling waves can be accessed, for a “file” wave search method with starting points indicated by the large black dots. Note in particular that in Figure 3.13c there is a small region in the lower left hand part of the parameter plane in which waves exist, but which cannot be accessed from the given starting point. As a useful rule of thumb, if the commands `ptw_loop` or `stability_loop` are run using the “file” wave search method, and if the detected region of periodic travelling waves has a boundary parallel to one of the axes, then this probably results from a limitation imposed by the selected starting

point, rather than being the actual boundary of the region of periodic travelling waves.

An important warning must be given concerning the value of the initial step size `ds` (specified in `constants.input`) when using the indirect or file methods. In these cases, calculation of the required periodic travelling wave usually involves two separate continuation stages, one varying the wave speed with a constant control parameter value, and the other varying the control parameter with a constant wave speed value. However, if the control parameter wave speed value for the required wave is the same as that specified in `equations.input` for the Hopf bifurcation search (indirect method) or for the solution in `ptwsoln.input` (file method), then WAVETRAIN omits the continuation stage in which the control parameter / wave speed is varied. In fact, it is omitted even if the two control parameter / wave speed values differ, but only slightly; the threshold difference for doing the continuation is determined by the constants `controlatol` and `controlrtol` / `speedatol` and `speedrtol`; these are set in `defaults.input`. Note that if the difference between the two control parameter / wave speed values is small but exceeds the threshold, then WAVETRAIN may fail to find the required periodic travelling wave because the first continuation step is too large, causing the numerical continuation to “jump over” the required point. The remedy for this is to reduce the initial continuation step size, `ds`, and also the minimum step size `dsmin` if this is necessary in order that $ds \geq dsmin$. The constants `ds` and `dsmin` are set in `constants.input`, and the relevant values are those in the second line of step size settings. It is recommended that relatively small values of these two constants are used in conjunction with the indirect or file wave search methods.

A single wave search method can be used for all control parameter and wave speed values, but the method and/or the Hopf bifurcation search range can also be parameter-dependent. Thus in order to cover the whole of the part of the control parameter – wave speed plane illustrated in Figure 3.11, the relevant part of `equations.input` could be:

```
# Wave search method and Hopf bifurcation search range
c<1 & P>1.5
  direct
  1.5
  1.8
c<1 & P<1.5
  direct
  1.5
  1.2
c>=1
  indirect : P=1.65
  0.5
  0.8
```

where `P` and `c` are the names for the control parameter and wave speed respectively, which would be set in `variables.input`. In this example all of the start and end values, and all of the inequality limits, are constant. However they can depend on the control parameter and wave speed, and also on the parameters defined in `other_parameters.input`. Note that although a mixed search method such as this can involve the “file” method, only one reference pair of control parameter and wave speed values can be used, since only one `ptwsoln.input` file is permitted.

An example illustrating the different search methods is provided in the problem `demo2`. This is identical to `demo` except for two changes. Firstly, the single wave search method

specified in `equations.input` for `demo` has been replaced by a more complicated mixed method, in which direct, indirect and file methods are used in different parts of the control parameter–wave speed plane. Secondly, the values of `ds` and `dsmin` for the continuation of periodic travelling waves have been reduced, as recommended on page 102 for the indirect and file methods. For the “file” case, a suitable wave solution has been provided in the `demo2` subdirectory of `input_files`. This solution was actually constructed from a WAVETRAIN output file, although in a real problem, output from a numerical simulation of the partial differential equations would normally be used. Entering the command

```
ptw_loop demo2
```

performs a loop across the parameter plane for this problem. One can plot the results in the usual way, by typing

```
plot demo2 101
```

(the number 101 is the pcode value) and then

```
@pplane
```

at the plotter prompt. If the default plot type is selected by entering RETURN (ENTER) when prompted, Figure 2.3 is reproduced. The plotter can then be exited in the usual way (@exit or @quit).

Finally, it is important to comment on the use of the parameter `iwave` with the different search methods. The setting `iwave=0` can only be used in the direct case. Settings of `iwave> 1` are unaffected by the choice of direct, indirect or file search method, with one exception. If a periodic travelling wave is required at a value of the control parameter that is the same as that specified for the Hopf bifurcation search in the indirect case, or for the control parameter value corresponding to the solution in `ptwsoln.input` in the file case, then the wave is calculated in a single continuation step, varying the wave speed: the continuation that would usually be done in the control parameter is not required. Therefore the calculation does not provide WAVETRAIN with a direction of travel along the periodic travelling wave solution branch for fixed wave speed. Hence settings of `iwave> 1` are not possible, and will cause WAVETRAIN to terminate the run with an error message.

3.1.13 Details of the Matrix Eigenvalue Calculation

The first step in determining periodic travelling wave stability is to calculate the eigenvalues corresponding to periodic eigenfunctions, which form starting points for numerical continuation of the eigenvalue spectrum. WAVETRAIN does this by replacing the derivatives (with respect to the travelling wave variable) in the eigenvalue equation by finite difference approximations, with periodic boundary conditions on a domain of length equal to one period of the wave. The order of these approximations is determined by the constant `order`, set in `constants.input` (see §2.2.1). The resulting matrix eigenvalue problem is solved using routines from version 3.1.1 of LAPACK (Anderson *et al.*, 1999), and the associated routines from BLAS (Lawson *et al.*, 1979; Dongarra *et al.*, 1988a,b; Dongarra *et al.*,

1990a,b). Further details of these packages are available from www.netlib.org/lapack and www.netlib.org/blas respectively. If the equations being studied are of the form (1.1a), meaning that each of equation contains a time derivative, then the eigenvalue problem is of standard type, and is solved using the LAPACK routine `dgeevx`. However if the equations are of the form (1.1b,c), meaning that there is no time derivative in one or more of the equations, then the eigenvalue problem is of generalised type, and is solved using the LAPACK routine `dggevx`. In fact, in this latter case it is possible to convert the eigenvalue problem into one of standard type (Sherratt, 2012), and future versions of WAVETRAIN may adopt this approach. However the coding changes required to use `dggevx` rather than `dgeevx` are very slight, and on these grounds WAVETRAIN currently adopts this less efficient approach.

LAPACK makes extensive use of the Basic Linear Algebra Subprograms (BLAS). To maximise computational speed when using LAPACK, one should use a BLAS library that is optimised for the particular computer being used. However, in the absence of such a library, there is a Fortran77 reference implementation of the BLAS, and this is supplied with WAVETRAIN. Users of WAVETRAIN are recommended to use this reference implementation unless they already have an optimised BLAS library, since LAPACK computations usually constitute only a small part of the run times for WAVETRAIN. However if users are interested in installing an optimised BLAS library, they should consult www.netlib.org/blas or the Wikipedia entry on BLAS. Once a user has their own BLAS library, it is incorporated into WAVETRAIN via the `blas` entry in `defaults.input` (see §3.2 for details).

Provision of a specific BLAS library by the user raises one small technical difficulty. There is a Fortran77 function `LSAME` that is used by both LAPACK and BLAS. The user's BLAS library may or may not include this function. If it does, then WAVETRAIN's version of `LSAME` must be excluded from the relevant Fortran77 compilations, and this is done via the `lsame` entry in `defaults.input` (see §3.2 for details).

Another issue of a similar type concerns the LAPACK function `DLAMCH`. This returns the value of “machine epsilon”, which gives a bound on the roundoff in individual floating-point operations. A version of this function is distributed with LAPACK (version 3.1.1), and this is supplied with WAVETRAIN. This function will work on the vast majority of computers, but not all. In the latter case, the user will have to provide their own version of `DLAMCH`; see the LAPACK documentation for information about this. A user-supplied version of `DLAMCH` can be incorporated into WAVETRAIN via provision of the file name (including full path) in the `dlamch` entry in `defaults.input` (see §3.2 for details).

The computed eigenvalues will differ from the true eigenvalues corresponding to periodic eigenfunctions for two separate reasons: error in the calculation of the matrix eigenvalues, and differences between the discretised and actual eigenfunction equations. The first of these is monitored by WAVETRAIN, which determines estimated error bounds for each of the calculated eigenvalues. It is important to clarify that these bounds refer to errors in the calculation of the matrix eigenvalues, rather than the errors of the calculated values as solutions of the actual eigenfunction equation. The estimated error bounds depend on whether the equations being studied are of the form (1.1a) or (1.1b,c). In the former case, the bound is simply the quantity `EERRBD` that is calculated by the relevant code fragment in the LAPACK user's guide (Anderson *et al.*, 1999), and is an approximate bound on the absolute value of the difference between the actual and computed matrix

eigenvalues. However the latter case, in which a generalised eigenvalue problem is solved, is somewhat more complicated. The corresponding code fragment in the LAPACK user guide provides a bound on the “chordal distance” between the actual eigenvalue (λ , say) and the computed eigenvalue ($\hat{\lambda}$, say),

$$\chi(\lambda, \hat{\lambda}) = \frac{|\lambda - \hat{\lambda}|}{(1 + |\lambda|^2)^{1/2} (1 + |\hat{\lambda}|^2)^{1/2}}.$$

For the purposes of WAVETRAIN, it is most useful to calculate instead a quantity that is an approximate bound on the error in the eigenvalue itself, and thus WAVETRAIN calculates

$$(1 + |\hat{\lambda}|^2) \chi(\lambda, \hat{\lambda}).$$

This will be a crude approximation to an error bound on the eigenvalue provided that the relative error is not too large, but users should be aware of its limitations. Of course, the user can back-calculate the chordal distance bound if desired, using WAVETRAIN’s crude estimate and the real and imaginary parts of the eigenvalue.

The error bounds for the matrix eigenvalues are written to the screen as part of the output from the `convergence_table` command, which is a vehicle for displaying the results of calculations done by the `eigenvalue_convergence` command (see §2.1.2). They are also written to the `info.txt` file for calculations of wave stability (see §2.5 and §4.4.2), and are listed in the file `evalues.data` in corresponding rcode output subdirectory (see §4.4.8).

One further important point concerning errors in the matrix eigenvalues is that the estimated error bounds are provided simply for information. Future versions of WAVETRAIN may provide alternative methods for calculating the matrix eigenvalues that could be used if the errors are too large, but this facility is not currently available.

The differences between the discretised and actual eigenfunction equations can usually be reduced simply by increasing the number of points in the discretisation, which is achieved by increasing `nmesh2`, which is set in the file `constants.input`. The command `eigenvalue_convergence` (see §2.1.2) facilitates the selection of a suitable value of `nmesh2`. Typically, as `nmesh2` increases the matrix eigenvalues converge to the functional eigenvalues. This is guaranteed by analytical convergence results in many cases (Atkinson, 1964; Kreiss, 1972; de Boor & Swartz, 1980; Chatelin, 1981), but for some equations more complicated behaviour can occur. An example of this can be obtained by editing the file `constants.input` for the problem `demo` as follows:

- change `nmesh1` to 160
- change `nevalues` to 20 (leaving `iposim=1`).

Having made these changes, one can enter the command

```
eigenvalue_convergence demo 2.2 0.7 50 100 150 200 250 300 350 400 450 500
```

(run time: about 2 hours on a typical desktop computer). Investigation of the results using the `convergence_table` command shows that the first four matrix eigenvalues converge in a straightforward manner to the corresponding functional eigenvalues. However for the fifth functional eigenvalue the situation is more complicated. There is a sequence of matrix eigenvalues that converges to this functional eigenvalue, as illustrated in Figure 3.14. However the appropriate matrix eigenvalue is only number 5 for relatively small `nmesh2` values. As `nmesh2` is increased above 100, additional matrix eigenvalues appear with real parts in between those of the fourth and fifth functional eigenvalues, and with very large imaginary parts ($\sim 10^4$). These eigenvalues appear to be an artifact of the discretisation. In a case such as this it is convenient to exclude matrix eigenvalues with very large imaginary parts from those used as starting points for numerical continuation of the spectrum. This can be achieved using the constant `evibound`, set in `defaults.input` (see §3.2).

Even when the matrix eigenvalues converge in a simple manner to the functional eigenvalues, it is typical that the numerical error bounds on their calculated values increases. This is because the condition number of the matrix increases as one of its eigenvalues approaches zero, which is always a functional eigenvalue.

An appropriate choice of `nmesh2` will be a compromise between these various trends, with the smallest viable choice being optimal to reduce run times. Moreover in difficult cases, setting the constant `order` to a value greater than one may enable a smaller value of `nmesh2` to be used; examples of this are given in §2.3 and at the end of §3.1.14.

Finally it is important to note that differences between the matrix and functional eigenvalues, whatever their cause, do not translate at all into errors in the calculated eigenvalue spectrum. All that is required is that the calculated matrix eigenvalues are sufficiently close to the functional eigenvalues that they provide a starting point from which numerical continuation can proceed. In reality the numerical continuation is typically very tolerant of poor starting approximations.

3.1.14 An Outline of How the `stability_boundary` Command Works

Although its syntax is not particularly complicated, the internal workings of the command `stability_boundary` are much more complex than those of the other WAVETRAIN commands. Of course the user does not need to be concerned with this, but it is helpful to understand the basic way in which the command operates, in order to make sense of the messages that are written to the screen and/or the `info.txt` file during a run. The description in this section will be deliberately brief: full details of the computational algorithms are given in Rademacher *et al* (2007) and Sherratt (2013b).

A brief summary of the mathematical background will be helpful to many users. Stability changes of periodic travelling waves can be of either Eckhaus or Hopf type (Rademacher & Scheel, 2007). Recall that the eigenvalue spectrum of a periodic travelling wave always passes through the origin, corresponding to neutral stability to translations. Also, the spectrum will always be symmetric about the real axis. A change of stability of Eckhaus type occurs when the curvature of the spectrum at the origin changes sign (illustrated schematically in Figure 3.15a). This was the type of stability change that occurred in

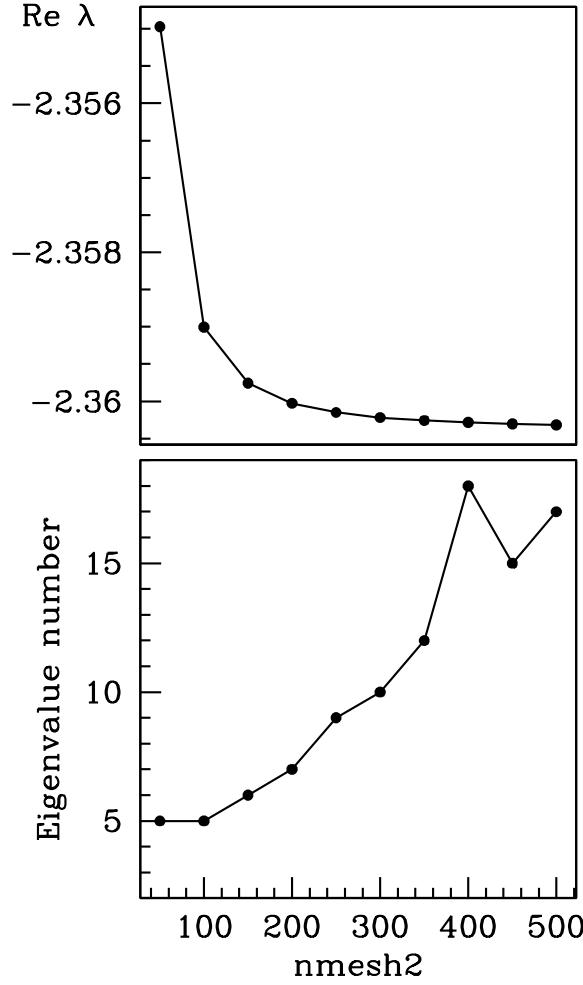


Figure 3.14: An example of a complicated pattern of convergence to a functional eigenvalue of the eigenvalues of the matrix given by discretising the eigenfunction equation in the travelling wave coordinate. This figure was not itself generated by WAVETRAIN, but the data that is plotted was generated by the `eigenvalue_convergence` command for an amended version of the problem `demo`, as described in the main text. After the run of this command, the command `convergence_table` was run for each of the eigenvalues numbered 5-18 (individually). In each case the table was examined for eigenvalues close to $-2.3603 + i * 0.9935$ (real and imaginary parts both within 0.1 would be a suitable criterion). Any such eigenvalues are plotted against the corresponding `nmesh2` value in the upper panel, and the corresponding eigenvalue number(s) are plotted against `nmesh2` in the lower panel. The results show that as the discretisation becomes finer, new matrix eigenvalues with real parts between those of the fourth and fifth functional eigenvalues appear; these eigenvalues all have very large imaginary parts ($\sim 10^4$) and appear to be an artifact of the discretisation.

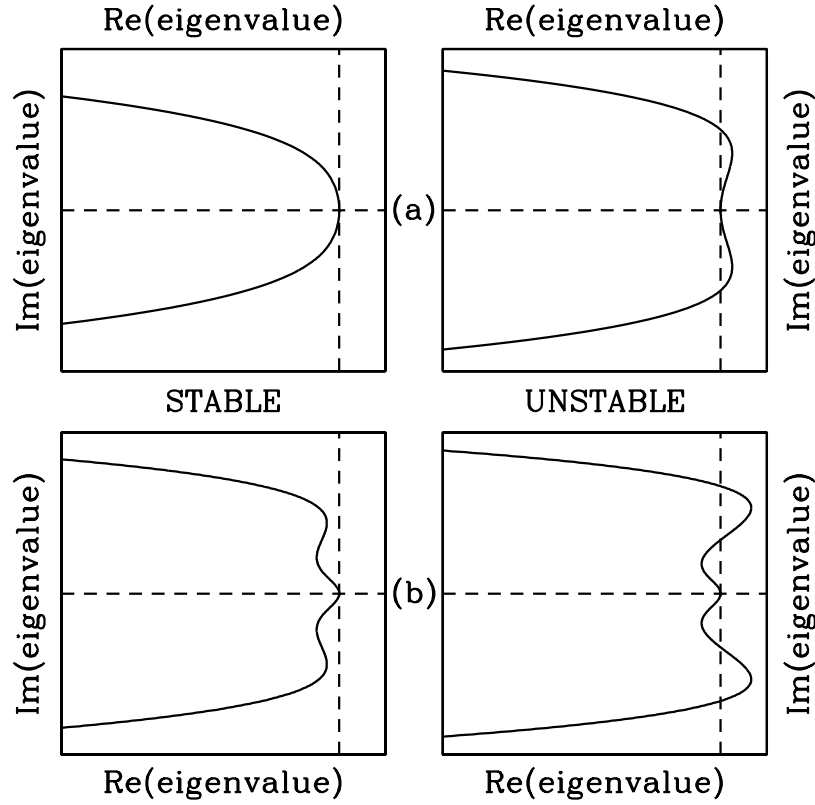


Figure 3.15: An illustration of the two different ways in which the stability of a periodic travelling wave can change. (a) and (b) show typical forms of the eigenvalue spectra on the stable (left) and unstable (right) side of a change in stability of (a) Eckhaus type and (b) Hopf type. The dashed lines denote the zero axes of the real and imaginary parts of the eigenvalue.

the problem `demo`, discussed in §2.1. The alternative possibility is that the spectrum passes through the imaginary axis away from the origin (illustrated schematically in Figure 3.15b). This type of stability change is referred to as being of Hopf type.

Recall that the `stability_boundary` command has six compulsory arguments, which are the input subdirectory name and the pcode value, followed by two pairs of control parameter and wave speed values; these may be followed by optional arguments (see §3.1.2). The command operates via the following stages.

Stage 1: Calculation of the periodic travelling wave for the control parameter and wave speed given in arguments 3 and 4. The value of `iwave` is used to determine which wave solution is being considered, in the case of multiple solutions (see §3.1.9).

Stage 2: Calculation of the eigenfunction corresponding to the zero eigenvalue, again for control parameter and wave speed given in arguments 3 and 4. `WAVETRAIN` does this by calculating the eigenvalues corresponding to periodic eigenfunctions via dis-

cretisation (see §3.1.13). The matrix eigenvalue with smallest modulus is interpreted as an approximation to the zero functional eigenvalue. Using this approximation as a starting point, WAVETRAIN performs a numerical continuation in a dummy parameter, which generates a numerical solution of the eigenfunction equation.

Stage 3: Calculation of the first and second derivatives of the eigenvalue and eigenfunction with respect to γ for the zero eigenvalue, again for control parameter and wave speed given in arguments 3 and 4. Here, as previously, γ is the phase difference in the eigenfunction across one period of the periodic travelling wave.

Stage 4: Numerical continuation of the periodic travelling wave, the eigenfunction for the zero eigenvalue, and the first and second derivatives of this eigenvalue and eigenfunction with respect to γ , as the control parameter/wave speed is varied between the value in arguments 3/4 and the value in arguments 5/6, with the wave speed/control parameter fixed at the common value of arguments 4 and 6/3 and 5. (Recall that either arguments 3 and 5 or arguments 4 and 6 must be the same.) During this continuation, WAVETRAIN monitors the value of the real part of the second derivative of the eigenvalue with respect to γ , looking for a zero, which would correspond to a change of stability of Eckhaus type. If such a stability change is detected, the command progresses to Stage 5A; if not, the next step is Stage 5B.

Stage 5A: If a zero of the real part of the second derivative of the zero eigenvalue with respect to γ is detected during Stage 4, then this zero point is continued numerically in the control parameter – wave speed plane, thereby tracing out the stability boundary. Since the initial zero point is in the interior of the region of the control parameter – wave speed plane under consideration, this continuation must be performed twice, with opposite initial directions. The command then ends.

Stage 5B: If a zero of the second derivative of the zero eigenvalue with respect to γ is not detected during Stage 4, WAVETRAIN concludes that the change of stability is of Hopf rather than Eckhaus type. WAVETRAIN then calculates the eigenvalue spectrum of the periodic travelling wave with control parameter and wave speed values given in arguments 3 and 4. Note that the eigenvalues with $\gamma = 0$, which form the starting point of this calculation, have been calculated previously in Stage 2.

Stage 6: Provided that the control parameter and wave speed values are sufficiently close to a stability change of Hopf type, the eigenvalue spectrum will contain at least one point with strictly positive imaginary part at which there is a fold, meaning that the slope of the spectrum is infinite (see Figure 3.15b). WAVETRAIN estimates the approximate location of the fold with positive imaginary part and largest real part via interpolation along the calculated spectrum. It then refines this estimate via numerical continuation of the equations for the periodic travelling wave, the eigenfunction, and the first derivative of the eigenvalue and eigenfunction with respect to γ . This relatively time-consuming continuation is performed over a small region of the spectrum containing the estimated fold location, and the size of this small region is determined by the constant `gatol`, set in `defaults.input`.

Stage 7: The fold located in Stage 6 is tracked numerically as the control parameter/wave speed is varied between the values in arguments 3/4 and 5/6, with the wave speed/control parameter fixed at the common value of arguments 4 and 6 / 3 and 5. (Recall that either arguments 3 and 5 or arguments 4 and 6 must be the same.) During this continuation, WAVETRAIN monitors the real part of the eigenvalue, looking for a point at which it is zero; this corresponds to a stability change of Hopf type, and will occur provided the initial control parameter and wave speed values are sufficiently close to the stability change.

Stage 8: The zero of the real part of the eigenvalue is continued numerically in the control parameter – wave speed plane, with its derivative with respect to γ also fixed at zero. This traces out the stability boundary. Since the initial zero point is in the interior of the region of the control parameter – wave speed plane under consideration, this continuation must be performed twice, with opposite initial directions.

At the end of Stage 8 the command ends. However, if a problem occurs during any of Stages 5B–8, WAVETRAIN calculates the periodic travelling wave and eigenvalue spectrum for the control parameter and wave speed values given in arguments 5 and 6, and then repeats Stages 6–8. This repetition is useful in case the algorithm works for the point given by arguments 5 and 6, but fails for the point given by arguments 3 and 4 because it is too far from the stability boundary. For example, there may not be a fold in the spectrum, other than zero, if one is sufficiently far from the stability boundary on the stable side.

A simple example of a change in stability of Hopf type is discussed by Bordiougov & Engel (2006); their equations are

$$\begin{aligned}\partial u / \partial t &= \partial^2 u / \partial x^2 + (1/\epsilon) [u - u^2 - (fv + \phi)(u - q)/(u + q)] \\ \partial v / \partial t &= u - v.\end{aligned}$$

The **demo3** input subdirectory contains files that enable these equations to be studied using WAVETRAIN. The parameter ϕ is used as control parameter, and its numerical value is quite small (about 10^{-4}) compared to the wave speed and the solution components. A size difference of this type is not recommended (see page 40), and it would be sensible to reformulate the equations using the parameter $\hat{\phi} = 10^4 \phi$. However, for the runs performed in this demonstration the size difference does not cause problems, and I deliberately do not rescale ϕ in order to illustrate the input of parameter values in scientific notation.

One important difference between **demo3** and the previous problems **demo**, **demo1** and **demo2** is that there is no algebraic formula for the homogeneous steady state from which periodic travelling waves arise. Therefore the input files must be augmented by a file **steadystate.input**, which lists the steady state values of the travelling wave variables at a series of values of the control parameter. This file must be generated by some external software, and for **demo3** this was done by the Fortran77 program **create_stst_file.f**; for the user's information, this program is provided in the **input_files/demo3** directory, but there is no need to run it since the **steadystate.input** file is also provided. In the part of the **equations.input** file corresponding to the steady states, one simply puts the word **file** (or **File** or **FILE**), and WAVETRAIN will then use **steadystate.input**. The command

`stability_loop demo3`

(run time: about six hours on a typical desktop computer) scans a region of the control parameter – wave speed plane. During the run, a warning about the value of `controlatol` appears for each pair of control parameter and wave speed values considered. This is because of the small numerical values of the control parameter in `demo3`. These warnings could be avoided either by rescaling ϕ as discussed above, or by decreasing `controlatol` (set in `defaults.input`). However the current settings do not cause any problems in practice and for demonstration purposes it is more convenient to retain them. Since this is the first run for `demo3`, it will be allocated the pcode value 101, and one can plot the results by typing

```
plot demo3 101
```

followed by the plotter command `@pplane`, selecting the default (`rcode`) option for the plot style. The result is illustrated in Figure 3.16a and reveals both stable and unstable periodic travelling waves. For example, the periodic travelling wave for $\phi = 8.21 \times 10^{-4}$, $c = 9.75$ is stable, while that for $\phi = 8.22 \times 10^{-4}$, $c = 9.75$ is unstable, and these points provide good starting points for tracing the stability boundary. To do this, one exits the plotter by typing `@exit` or `@quit`. One then runs the command

```
stability_boundary demo3 101 8.21E-4 9.75 8.22E-4 9.75
```

which moves progressively through each of the 8 stages described above (run time: about 90 minutes on a typical desktop computer). Various messages appear on the screen during the run to update the user on the progress of the calculation. Included amongst these is a warning about a small number of continuation steps, which results from the point $\phi = 8.21 \times 10^{-4}$, $c = 9.75$ being very close to the stability boundary. The warning about the value of `controlatol` also appears. Both of these warnings can be ignored.

To plot the result of this calculation, one re-enters the plotter by typing

```
plot demo3 101
```

and then regenerates the results of the `stability_loop` run via the plotter command `@pplane`, in this case selecting the “symbol” plot style for variety. The stability boundary can then be superimposed on this plot by typing

```
@stability_boundary 101
```

giving the result illustrated in Figure 3.16b. The colour of the stability boundary shows that it is of Hopf rather than Eckhaus type (see Figure 2.4). To see the change in the eigenvalue spectrum as the boundary is crossed, one can type `@spectrum 1003` and then `@spectrum 1002`, giving the eigenvalue spectra illustrated in Figures 3.17a,b. This demonstrates clearly that the change in stability arises from the spectrum crossing the imaginary axis away from the origin. The plotter can be exited by typing `@exit` or `@quit`. Another example of a stability boundary of Hopf type is given in Sherratt (2013a): the equations considered there are a model for pattern formation in mussel beds.

One important warning about the `stability_boundary` must be given, concerning step size in the case of a change in stability of Hopf type. Any numerical continuation

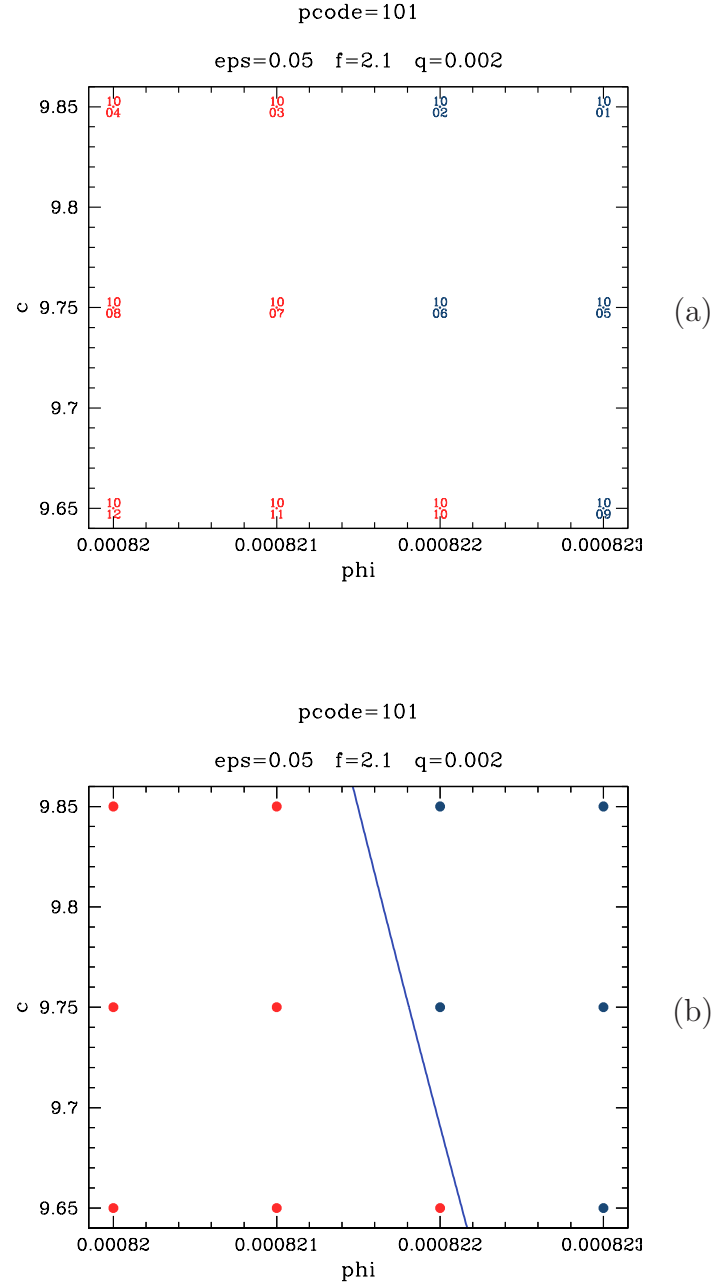


Figure 3.16: Results on the stability of periodic travelling wave solutions for the problem `demo3`. (a) The results of a loop over a grid of points in the control parameter – wave speed plane. (b) The same results with the boundary between stable and unstable waves superimposed. The colour of this boundary shows that the stability change is of Hopf type (see Figure 2.4). The run and plot commands used to generate these figures are given in the main text, and are also listed in the Appendix. The quantities `eps`, `f` and `q` referred to in the second line of the title are parameters which appear in the equations but which are not being used as the control parameter; their values are specified in the file `other_parameters.input` (see §2.2.3).

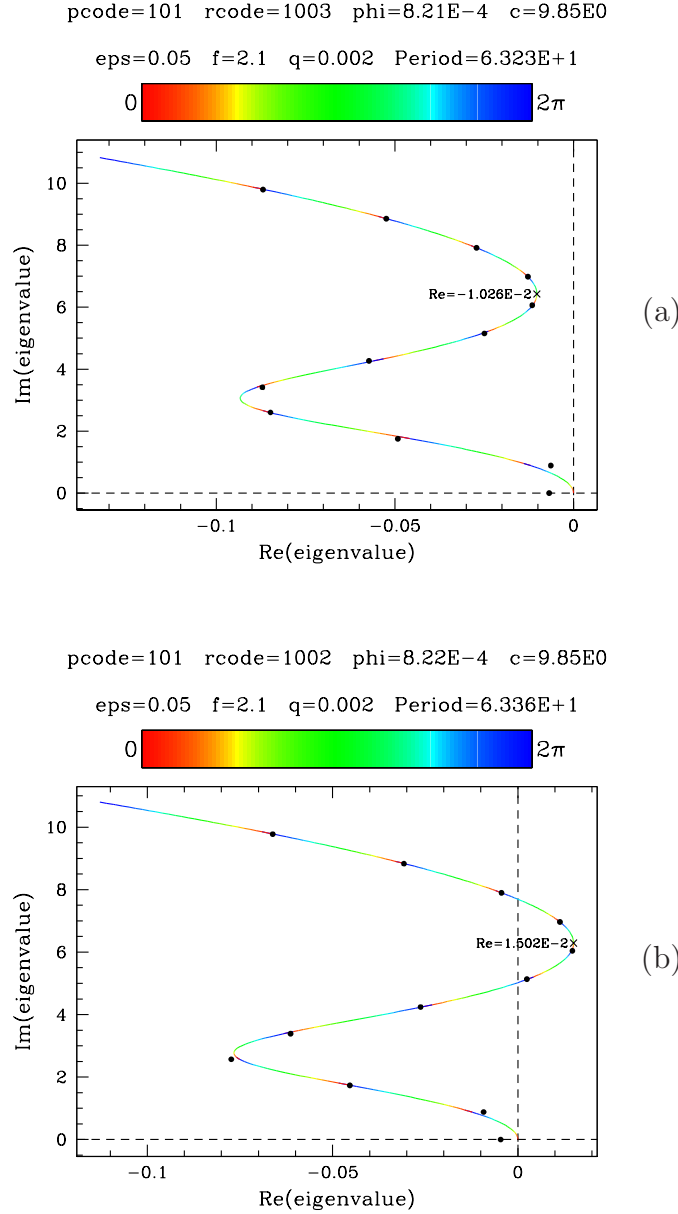


Figure 3.17: Eigenvalue spectra of periodic travelling wave solutions for the problem `demo3`. The control parameter values in (a) and (b) are such that the wave is stable and unstable, respectively. These figures illustrate that the onset of instability is associated with the spectrum crossing the imaginary axis away from the origin, i.e. it is of Hopf type. The dots indicate the (rather poor) approximations to eigenvalues corresponding to periodic eigenfunctions, calculated by discretising the eigenfunction equation, and the colours along the curves indicate the phase difference in the eigenfunction over one period of the periodic travelling wave. The run and plot commands used to generate these figures are given in the main text, and are also listed in the Appendix. The quantities `eps`, `f` and `q` referred to in the second line of the title are parameters which appear in the equations but which are not being used as the control parameter; their values are specified in the file `other_parameters.input` (see §2.2.3).

calculation requires that the step size be sufficiently small in order for the calculation to be successful. There is no *a priori* means of knowing how small will be “sufficiently small” in a particular case. Numerical continuation of a stability boundary of Hopf type is defined by the conditions that the real part of the eigenvalue, and the real part of its derivative with respect to γ , are both zero. As well as holding (away from the origin) at a change in stability of Hopf type, these conditions both hold at the origin for the spectrum of any periodic travelling wave. Therefore if too large a step size is used, the continuation step can erroneously locate the origin, or a point close to it. Once the continuation has veered off in this way, the boundary-tracking algorithm will trace out a more or less random path in the parameter plane. In view of this it is important to check that the stability boundary remains on track throughout the calculation. This can be done either via a run of `stability_loop`, or by using `stability` to calculate the spectrum at a series of points along the stability boundary. In the latter case, appropriate pairs of control parameter and wave speed values can be obtained either via the optional arguments to the `stability_boundary` command (see §3.1.2) or by inspection of the output file `stabilityboundary.boundarydata` in the relevant scode output directory (see §4.4.9). If the calculation has veered off, the remedy is to reduce the step sizes `ds`, `dsmin` and `dsmax` (set on line 9 of `constants.input`) and rerun the `stability_boundary` command.

To conclude this section, I comment on a difference between `demo3` and `demo`, `demo1` and the previous problems `demo2`, namely that the constant `order` (set in `constants.input`) is set to 2 rather than 1. This constant specifies the order of the finite difference approximation used for the highest derivative when discretising the eigenfunction equation. Thus setting `order` = 1 causes WAVETRAIN to use a three-point approximation for the second derivative, while `order` = 2 gives a five-point approximation. In simple cases one would certainly use `order` = 1, but the higher level of accuracy resulting from `order` > 1 can be useful when convergence of the eigenvalues is slow as the number of mesh points (determined by `nmesh2`) is increased. The problem `demo3` provides an example of this. Figure 3.18 shows output from the `convergence_table` command after running `eigenvalue_convergence` with a range of `nmesh2` values, for typical values of the control parameter and wave speed. The upper two convergence tables show the results obtained with `order`= 1 for eigenvalues 9 and 2, while the lower two tables show the corresponding results for `order`= 2. It is typical that as `nmesh2` is increased, the error bounds on the eigenvalues increases, and there is a corresponding increase when `order` is increased for fixed `nmesh2`. Both of these trends are clear in Figure 3.18, for both of the eigenvalues. These trends occur because for the matrix arising from discretising the eigenfunction equation, the condition number increases as the discretisation becomes a better approximation to the eigenfunction equation, which has a zero eigenvalue. Considering first eigenvalue 9 for `order`= 1, the results in Figure 3.18 strongly suggest convergence to an eigenvalue of about $-0.067 + 8.8i$ as `nmesh2` is increased. For `order`= 2, the eigenvalue approaches this value more rapidly as `nmesh2` is increased, but then moves away as it becomes dominated by numerical error. For eigenvalue 2, the pattern is the same, but the error bounds are much more significant relative to the real part of the eigenvalue.

When choosing values for `nmesh2` and `order`, it must be remembered that any inaccuracies in the eigenvalue approximations do not lead to inaccuracies in the computed spectrum – all that is required is that the approximations are sufficiently good that they

(a) order=1

nmesh	Re of eval	Im of eval	Error bound
100	-0.42201304E-01	0.24051920E+01	0.55334430E-04
200	-0.71010858E-01	0.86053281E+01	0.34546716E-05
400	-0.65037266E-01	0.87937590E+01	0.11489065E-04
800	-0.66672646E-01	0.88431720E+01	0.39065761E-04
1200	-0.67058826E-01	0.88524301E+01	0.11296455E-03

nmesh	Re of eval	Im of eval	Error bound
100	0.10474546E+00	0.49714409E+01	0.44593174E-04
200	0.24245619E-01	0.59628161E+01	0.12892151E-04
400	-0.81155430E-02	0.60342199E+01	0.83242802E-04
800	-0.12355648E-01	0.88670188E+00	0.18454233E-01
1200	-0.11838610E-01	0.88681645E+00	0.41529717E-01

(b) order=2

nmesh	Re of eval	Im of eval	Error bound
100	-0.85883103E-01	0.87211437E+01	0.22435235E-01
200	-0.68847449E-01	0.88493459E+01	0.25447809E-02
400	-0.67493922E-01	0.88591701E+01	0.22255001E-02
800	-0.67363382E-01	0.88598102E+01	0.16993811E-01
1200	-0.73503679E-01	0.17237650E+01	0.20880794E+02

nmesh	Re of eval	Im of eval	Error bound
100	-0.41847151E-02	0.00000000E+00	0.12590945E+00
200	-0.12490974E-01	0.88671728E+00	0.59945934E+00
400	-0.15447779E-01	0.88308056E+00	0.25714288E+01
800	-0.11372513E-01	0.88860381E+00	0.10258903E+02
1200	-0.22987593E-01	0.60612471E+01	0.10648373E+01

Figure 3.18: The change in the eigenvalues of the discretised eigenfunction equations as the constant `nmesh2` (defined in `constants.input`) is varied. In (a) the parameter `order` (also defined in `constants.input`) has been changed to 1; in (b) it has been reset to 2. The run command for these tables is `eigenvalue_convergence demo3 8.21E-4 9.75 100 200 400 800 1200` (the same for both (a) and (b); the only difference is the value of `order`). These commands are very time-consuming because of the large values of `nmesh2`: run times are about 15 hours for (a) and 20 hours for (b) on a typical desktop computer. Assuming that these are the first two runs of the `eigenvalue_convergence` command for `demo3`, they will be allocated `ecode` values of 101 and 102. The tables shown can then be generated via the commands `convergence_table demo3 101 9` and `convergence_table demo3 101 2` for (a), and `convergence_table demo3 102 9` and `convergence_table demo3 102 2` for (b).

converge to the points with $\gamma = 0$ on the computed spectrum. In the `constants.input` file for `demo3`, the settings `nmesh2=400` and `order = 2` are used as a compromise between accuracy and run time, and all of the 12 eigenvalues considered do converge to the points with $\gamma = 0$ on the computed spectrum, for all of the parameter sets considered. Here the number 12 is the value of `nevalues`, set in `constants.input`. Note that for publication or presentation purposes it would clearly be best to omit the dots showing the eigenvalues corresponding to periodic eigenfunctions in Figure 3.17, and this can be achieved by changing the plotter setting `showperiodiceigenvalues` to `no` in `plot_defaults.input` (see §3.4.1).

3.1.15 Multiple Simultaneous Runs of WAVETRAIN

The WAVETRAIN commands create and delete a number of temporary files during execution. Consequently it is not possible to run more than one command simultaneously, since these files might get deleted or overwritten in an inappropriate way. Such simultaneous runs are prevented by the creation of the file `lockfile` at the start of a command, which is deleted when the command finishes. Specifically, the following WAVETRAIN commands are protected via the creation of `lockfile`: `add_points_list`, `add_points_loop`, `bifurcation_diagram`, `cleanup`, `eigenvalue_convergence`, `hopf_locus`, `new_pcode`, `period_contour`, `ptw`, `ptw_loop`, `stability`, `stability_boundary`, `stability_loop`. Any other command can be used while one of these protected commands is being run. If a user wishes to run two of the protected WAVETRAIN commands in parallel, this is possible by installing two copies of WAVETRAIN in different parent directories. The two copies will be entirely decoupled, and plotting will then also have to be done separately in the two main directories.

In the same way, two simultaneous runs of the `plot` command are not possible because shell scripts are run in preparation for plotting that create various temporary files. Such simultaneous runs are prevented by the creation of the file `plot_lockfile` when the `plot` command is run; this file is deleted by the `@quit` or `@exit` commands.

However WAVETRAIN plotting can be done at the same time as new runs: there is no conflict regarding temporary files. Note that although the WAVETRAIN plotter does of course use the `plot_defaults` and `.style` input files, it does not require any of the run input files, which can therefore be changed as desired between runs. Moreover, note that the output subdirectory for a WAVETRAIN run contains a copy of all of the run input files as they were when the command was run, providing a full record of the run (see §4.4 for details).

3.2 Details of defaults.input

This file is located in the main `input_files` directory, and contains the values of computational constants that the user will only rarely want to change. This file complements the `constants.input` files in subdirectories of `input_files`, which contain the constants that are expected to vary between problems. Apart from comment lines, the version of the file provided at installation is as follows.

```

0.00001 0.001 # controlatol controlrtol
0.00001 0.001 # speedatol speedrtol
0.1 0.05 # periodatol periodrtol
9999 9999 500 9999 9999 10 # nmx:  hb, ptw, bd, spectrum, sb, dummy
0.0 100.0 # a0 a1
1.0E-6 1.0E-6 1.0E-4 # epsl epsu epss for Hopf bifn & ptw continuation
1.0E-5 1.0E-5 1.0E-3 # epsl epsu epss for spectrum & stability bdy
0.0001 -1 # evzero evibound
0.1 # gextra: overlap between plotted segments of the spectrum
1 # help display (1/2=scroll/list)
1.0E-8 # tolss: rel error tol on ptw variables for num soln of st st
200 300 # nwarn1 nwarn2: thresholds for warnings about number of steps
0.5 # grange: phase diff interval over which to search for fold
0.001 # gatol: abs tol for phase diff, used only for Hopf stab bdy
f77 # fortran77 compiler, can include option flags
    # blas:  blank, or name of user version (incl full path)
y # lsame:  y/n = use/do not use the wavetrain version
    # dlamch: blank, or name of user version (incl full path)

```

The meaning of the various constants is as follows.

controlatol, controlrtol

After the continuation of the periodic travelling wave solutions finishes, the tolerances `controlatol` and `controlrtol` are used to assess whether the required value of the control parameter has been reached. The criterion is that the absolute value of $(\text{control_parameter_soln} - \text{control_parameter_target})$ be less than $\text{abs}(\text{controlatol}) + \text{abs}(\text{controlrtol} * \text{control_parameter_target})$. These tolerances are needed because in AUTO a successful solution output at the specified value of the control parameter will not (necessarily) be exactly at the target value: AUTO has its own tolerance, which is determined by `epss`. However it is not feasible for WAVETRAIN to use the same tolerance criterion as AUTO because the AUTO criterion is based on continuation arc length. If the target value of the control parameter is assessed as not having been reached when in reality AUTO has reached it successfully, then either `controlatol` and/or `controlrtol` should be made smaller, or `epss` (for ptw continuation) should be made larger.

speedatol, speedrtol

The tolerances `speedatol` and `speedrtol` are analogous to `controlatol` and `controlrtol`, but for the wave speed. When calculating Hopf bifurcation loci, fold loci and period contours, continuation is sometimes done using the wave speed as the main continuation parameter. In such cases, the tolerances `speedatol` and `speedrtol` are used to assess whether the end points of the continuation range have been reached, via the criterion that $\text{abs}(\text{speed} - \text{speed_end})$ must be less than the critical value $\text{abs}(\text{speedatol}) + \text{abs}(\text{speedrtol} * \text{speed_end})$ in order for the end value `speed_end` to be assessed as having been reached. These tolerances are needed because in AUTO a successful solution output at the specified value of the wave speed will not (necessarily) be exactly at the target value: AUTO has its own tolerance, which is determined by `epss`. However it is not feasible for WAVETRAIN to use the same tolerance criterion as AUTO because the AUTO criterion is based on continua-

tion arc length. If the target value of the wave speed is assessed as not having been reached when in reality AUTO has reached it successfully, then either `speedatol` and/or `speedrtol` should be made smaller, or `epss` (for ptw continuation) should be made larger.

`periodatol, periodrtol`

One of the methods of calculating period contours involves specifying the required period. In this case, the tolerances `periodatol` and `periodrtol` are used to assess whether the required period has been reached. The criterion is that the absolute value of $(\text{period} - \text{period_target})$ be less than $\text{abs}(\text{periodatol}) + \text{abs}(\text{periodrtol} * \text{period_target})$. These tolerances are needed because in AUTO a successful solution output at the specified value of period will not (necessarily) be exactly at `period_target`: AUTO has its own tolerance, which is determined by `epss`. However it is not feasible for WAVETRAIN to use the same tolerance criterion as AUTO because the AUTO criterion is based on continuation arc length. If WAVETRAIN assesses that the target value of period has not been reached when in reality AUTO has reached it successfully, then either `periodatol` and/or `periodrtol` should be made smaller, or `epss` (for ptw continuation) should be made larger.

`nmx`

The `nmx` values specify the maximum number of continuation steps used in the various numerical continuation calculations.

`nmx_hb` is used when locating the Hopf bifurcation in the periodic travelling wave equations.

`nmx_ptw` is used for continuation of branches of periodic travelling wave solutions, except in bifurcation diagram calculations. It is used in the calculation of contours of constant period. It is also used for tracing the loci of Hopf bifurcation points and folds in the control parameter–wave speed plane (though in the former case the starting point on the locus is found using `nmx_hb`).

`nmx_bd` is used for the continuation of branches of periodic travelling wave solutions in bifurcation diagrams.

`nmx_spectrum` is used for continuation along the eigenvalue spectrum.

`nmx_sb` is used for locating and continuing stability boundaries of both Eckhaus and Hopf type.

`nmx_dummy` is used for all continuations in dummy parameters. Such continuations occur as an initial step in the calculation of the eigenvalue spectrum and of stability boundaries and in order to determine a starting solution for derivatives of the eigenfunction. In the calculations of stability boundaries and of periodic travelling waves when using an indirect search method, there are some short continuations used to test the continuation direction, and `nmx_dummy` is also used for these.

For dummy parameter continuations, the `nmx` value will determine the end point of the continuation and a small value is recommended; however `nmx_dummy` should

not be too small because of its use in the assessment of continuation directions; the recommended value is 10. For `nm_x_hb`, `nm_x_ptw` and `nm_x_spectrum`, the setting should be large enough that computations do not end because the number of steps reaches `nm_x`, except perhaps in the case of an error in the values of `ds`, `dsmin` or `dsmax` (given in `constants.input`). The recommended settings are 9999, simply because AUTO output (which appears in the `info.txt` file, and on the screen if the constant `info` is set to 4) can be confusing when step numbers have more than four digits: the values can be increased above 9999 should this prove necessary.

The constant `nm_x_bd` should typically not be set to such a large number. Bifurcation diagram continuations stop because either limits on the continuation parameter (which is either the control parameter or the wave speed) or on the solution amplitude are exceeded, or because the maximum number of continuation steps is reached, or because of a convergence error (which often indicates that a homoclinic solution is being approached). This contrasts with other continuations done in WAVETRAIN, for which there is one or more target values of the continuation parameter at which the continuation ends. The appropriate value should be large enough to allow the solution branch to approach the homoclinic solution when this is the end point, but small enough that the run time is not excessive if there is some other end point (for example the solution branch might ping-pong between two different Hopf bifurcations, in which case AUTO will typically compute backwards and forwards repeatedly along the same solution branch).

`a0`, `a1`

The constants `a0` and `a1` are the lower and upper limits on the solution amplitude: continuation will stop if this is exceeded. The same values are used in all AUTO runs. Suitably small and large values respectively are recommended: it is not expected that continuation will ever stop because these limits are reached, during normal execution. Note that the L2-norm is used for the solution amplitude in all AUTO runs (this corresponds to setting the AUTO parameter `iplt` to 0).

`eps1`, `epsu`, `epss`

The constants `eps1`, `epsu` and `epss` are tolerance parameters for convergence criteria during AUTO runs. The constants are exactly as used in AUTO: `eps1` and `epsu` determine the relative convergence criteria for equation parameters and solution components respectively, in the Newton/Chord method used by AUTO, while `epss` determines the relative convergence criterion for the detection of special solutions. The recommended ranges are 10^{-6} to 10^{-7} for `eps1` and `epsu`, and 10^{-4} to 10^{-5} for `epss`. (In particular, `epss` should be 100 to 1000 times larger than `eps1` and `epsu`). The first trio of values is used for locating Hopf bifurcations, for continuing branches of periodic solutions, and for calculating contours of constant period, bifurcation diagrams, and Hopf bifurcation loci. The second trio of values is used for spectrum continuations and for the calculation of stability boundaries of both Eckhaus and Hopf type. To speed up execution it is recommended that the second trio is larger than the first.

`evzero`

The constant **evzero** is used by the commands **stability** and **stability_loop**. These commands calculate the spectrum, and then loop round the points in the calculated spectrum, looking for folds. The periodic travelling wave is assessed as being unstable/stable according to whether there is/is not a fold away from the origin in the right hand half of the complex plane. The constant **evzero** is used to assess whether a fold is “away from the origin”: the criterion is that the amplitude of the fold location is greater than **evzero**. If **iposim**=1 then the constant **evzero** is also used in a different way by the commands **stability** and **stability_loop**, in tests of whether or not the setting **iposim**=1 is appropriate for the spectrum being calculated (see §3.1.8 for details). The constant **evzero** is also used by the **stability_boundary** command, which locates the fold away from the origin with largest real part as a precursor to detecting changes in stability of Hopf type.

evibound

The first stage in the calculation of the spectrum involves discretising the eigenfunction equation in the travelling wave coordinate, with periodic boundary conditions, and solving the resulting matrix eigenvalue problem. The matrix eigenvalues are ordered by the size of their real part, and a number **nevals** of them are used as starting points for continuation of the spectrum, with the constant **iposim** determining whether or not to include eigenvalues whose imaginary part is strictly negative; **nevals** and **iposim** are specified in **constants.input**. In some cases it is convenient to exclude from consideration matrix eigenvalues whose imaginary part exceeds a particular threshold in absolute value, and the constant **evibound** specifies this threshold. If **evibound** is negative, then no threshold is applied.

gextra

Let γ denote the phase difference in the eigenfunction across one period of the periodic travelling wave. Then the constant **gextra** determines the range of values of γ for which each spectrum segment is calculated: the range is $0 < \gamma < 2\pi + \text{gextra}$. This entire range is plotted by the plotter command **@spectrum**. There is no problem with setting **gextra** to zero, but a small non-zero value of **gextra** may help the visual appearance of the spectra, by preventing gaps between successive segments. Also, it might occasionally be useful to set **gextra** to a much larger value as a diagnostic tool. To ensure that $\gamma = 2\pi$ and $\gamma = 2\pi + \text{gextra}$ can be resolved as separate output points during continuation, **gextra** is interpreted as zero if it is less than $10 \times \text{dsmin}$.

help display

Output from the **wt_help** command can be displayed by either scrolling or listing the help file. The code number determines which of these display formats is used. Scrolling is accomplished via the **man** facility on the user’s system. Note that the help display format for the plotter’s **@help** command is set separately, in **plot_defaults.input**.

tolss

In general, there will not be an exact formula for the steady state, and therefore

this will have to be found numerically, using the user-supplied formulae/values as an initial guess. This numerical solution is done using the minpack routine `hybrd1` (Moré *et al.*, 1984), which uses a modification of the Powell hybrid method, and which is freely available from www.netlib.org. This routine gives a solution for which the estimated relative error in the solution is at most `tolss`.

`nwarn1, nwarn2`

Very large or very small numbers of continuation steps are not desirable at any stage of the calculation. A large number of steps typically means long run times, possibly unnecessarily. A small number of steps means that bifurcation points might be jumped over without detection. WAVETRAIN gives a warning message if the number of steps in any stage of the calculation is either very small or very large. The constants `nwarn1` and `nwarn2` (`nwarn1 < nwarn2`) are the thresholds for these warning messages to be given.

`grange`

When the change in periodic travelling wave stability is of Hopf (rather than Eckhaus) type, the first step in locating and tracking the stability boundary is to locate the point in the spectrum, away from the origin, at which there is a fold with the largest real part. This is calculated approximately when calculating the spectrum, in order to assess wave stability, but in order to calculate the stability boundary precisely a more accurate calculation must be done, via continuation of the equations for the first derivative of the eigenfunction (with respect to $i\gamma$) as well as the eigenfunction; here, as above, γ denotes the phase difference in the eigenfunction across one period of the periodic travelling wave. This can be a time-consuming calculation and therefore it is helpful to reduce the range of $i\gamma$ values over which it must be done. The constant `grange` specifies this range: if γ_{max} is the estimated fold location based on the spectrum calculation, then the range of γ values used to search for the exact location of the fold is $\gamma_{max} - \text{grange}/2 < \gamma < \gamma_{max} + \text{grange}/2$.

`gatol`

With γ , `grange` and `gmax` as described above, a preliminary stage in the calculation of a stability boundary of Hopf type is to continue the spectrum up to the point with $\gamma = \text{gmax} + \text{grange}/2$. The tolerance `gatol` is used to assess whether this end value of γ has been reached. Since it is known that γ takes values between roughly 0 and 2π , a single (absolute) tolerance is sufficient, and a relative tolerance is not required. Note that it is essential that `gatol` is significantly smaller than `grange`. If WAVETRAIN assesses that the target value of γ (which is $\text{gmax} + \text{grange}/2$) has not been reached when in reality AUTO has reached it successfully, then either `gatol` should be made smaller, or `epss` (for spectrum continuation) should be made larger.

`fortran77 compiler`

This is the name of the Fortran77 compiler to be used, e.g. `f77` or `g77`. It can be followed by one or more option flags. One reason for including a user-specified flag would be if by default the compiler outputs a list of subroutines during compilation. While this would not cause any errors for WAVETRAIN, it does interfere significantly

with screen output and therefore should be suppressed. The appropriate flag for this is compiler dependent, but `-silent` or `-fsilent` are likely possibilities. Note that the “-” symbol does need to be specified (e.g. `f77 -fsilent`).

blas

BLAS (Basic Linear Algebra Subprograms) are used extensively by LAPACK; in WAVETRAIN, LAPACK is used to calculate approximations to the eigenvalues and eigenfunctions for the case when the eigenfunctions are periodic over one period of the periodic travelling wave. WAVETRAIN is supplied with the Fortran77 reference implementation of the BLAS (specifically the BLAS files that are distributed with version 3.1.1 of LAPACK), and this will usually be adequate. However calculations done by LAPACK are significantly faster if one uses a machine-specific optimised BLAS library. Such libraries are freely available for many computer architectures: see www.netlib.org/blas for details. This entry in `defaults.input` provides the user with the opportunity to use such a machine-specific library, by giving the file name including full path. The file name can be either an uncompiled fortran (.f) file, or a compiled (.o) file, or a library (.a). In the third case, the entry must also include the library path, e.g. `-L/usr/lib -lmyblas` (for the library `/usr/lib/libmyblas.a`). Note that whatever the user enters on this line of `defaults.input` is used directly in the Fortran77 compilation command for the eigenvalue calculation program. Alternatively the entry can be blank, in which case WAVETRAIN will use the Fortran77 reference implementation of the BLAS. Typically the calculations done by LAPACK constitute only a relatively small part of the run times in WAVETRAIN and therefore it is recommended that the Fortran77 reference implementation of the BLAS be used unless the user already has a machine-specific optimised BLAS library.

lsame

The Fortran77 function `LSAME` is used by both BLAS and LAPACK. If the user is providing their own version of BLAS (see above) then this may or may not include the function `LSAME`. If it does then the version of `LSAME` supplied with WAVETRAIN should be excluded from compilation of the eigenvalue calculation program, by entering `n` in this line of `defaults.input`. If the user-supplied BLAS does not include `LSAME`, or if the Fortran77 reference implementation of the BLAS is being used, then `y` should be entered on this line, to indicate that the version of `LSAME` supplied with WAVETRAIN should be included in the compilation.

dlamch

The Fortran77 function `DLAMCH` is used by LAPACK, and returns the value of “machine epsilon”, which gives a bound on the roundoff in individual floating-point operations. A version of this function is distributed with LAPACK (version 3.1.1), and this is supplied with WAVETRAIN. This function will work on the vast majority of computers, but not all. In the latter case, the user can enter on this line of `defaults.input` the name (including full path) of a file containing an alternative version of `DLAMCH`. The file can be either an uncompiled (.f), or compiled (.o) file, or a library (.a) containing the function. In the third case, the entry must also include the library path, e.g. `-L/usr/lib -lmylibrary` (for the library

/usr/lib/libmylibrary.a). Note that whatever the user enters on this line of defaults.input is used directly in the Fortran77 compilation command for the eigenvalue calculation program. If the entry is blank, then the version of DLAMCH supplied with WAVETRAIN will be used.

3.3 Further Details of Plot Commands

3.3.1 Changing Plotter Styles

One of the objectives of WAVETRAIN is to produce graphical output suitable for publication. In many cases, users will wish to produce greyscale plots for this purpose, although during the research process they may find it more convenient to use colours to distinguish the different line and point types. To switch to greyscale mode, one enters the plot command

```
@set_style greyscale
```

and one can type

```
@set_style colour
```

to return to colour output. Note that the `@set_style` command creates a new postscript file of the parameter plane key, such as that illustrated in Figure 2.4 on page 18. Therefore if a postscript file is being written then it is ended by the `@set_style` command, with plotting returned to the screen after the command finishes.

Users may also wish to create their own plotter styles. To do this, one goes into the `input_files` directory and copies either `colour.style` or `greyscale.style` to a new file such as `my_settings.style`. The various settings in this file are discussed in §3.4; they cover the colour, thickness and style of all lines, plus point colours and types. The explanatory information in §3.4.2 is repeated in comments in the versions of `colour.style` and `greyscale.style` provided at installation; these comments describe in detail the meaning of each setting. The available line styles are explained in these comments. For colours, a total of 186 colours are available in WAVETRAIN, consisting of the 133 standard Crayola crayon colours, colours of the rainbow, primary and secondary colours (which are not all standard Crayola crayon colours), and 50 shades of grey, named `grey1` (almost black) up to `grey50` (almost white). A list of these colour names is provided as `colour_list.eps` in the `postscript_files` directory; this list is reproduced in Figure 3.19. Note that `colour_list.eps` is recreated each time the plotter is started. For details of Crayola crayon colours, see http://en.wikipedia.org/wiki/List_of_Crayola_crayon_colors.

3.3.2 Changing Other Plotter Settings

A variety of other plotter settings are contained in the file `plot_defaults.input` in the `input_files` directory. This includes the choice of plotter (`sm` or `gnuplot`), the specification of font sizes for text, the line and column spacings in the keys, the default choice of plotter style file, and the default choice between code numbers, symbols and

almond	greenblue	razzmatazz	grey08
antiquebrass	greenyellow	red	grey09
apricot	hotmagenta	redorange	grey10
aquamarine	inchworm	redviolet	grey11
asparagus	indigo	robinseggblue	grey12
atomictangerine	jazzberryjam	royalpurple	grey13
bananamanía	junglegreen	salmon	grey14
beaver	laserlemon	scarlet	grey15
bittersweet	lavender	screaminggreen	grey16
black	lemonyellow	seagreen	grey17
blizzardblue	macaroniandcheese	sepia	grey18
blue	magenta	shadow	grey19
bluebell	magicmint	shamrock	grey20
bluegreen	mahogany	shockingpink	grey21
bluegrey	maize	silver	grey22
blueviolet	manatee	skyblue	grey23
blush	mangotango	springgreen	grey24
brickred	maroon	sunglow	grey25
brown	mauveous	sunsetorange	grey26
burntorange	melon	tealblue	grey27
burntsienna	midnightblue	thistle	grey28
cadetblue	mountainmeadow	ticklempink	grey29
canary	mulberry	timberwolf	grey30
caribbeangreen	navyblue	tropicalrainforest	grey31
carnationpink	neoncarrot	tumbleweed	grey32
cerise	olivegreen	turquoiseblue	grey33
cerulean	orange	unmellowyellow	grey34
chestnut	orangered	violet	grey35
copper	orangeyellow	violetblue	grey36
cornflower	orchid	violetred	grey37
cottoncandy	outerspace	vididtangerine	grey38
crayolagray	outrageousorange	vividviolet	grey39
crayolatan	pacificblue	white	grey40
cyan	peach	wildblueyonder	grey41
dandelion	periwinkle	wildstrawberry	grey42
denim	piggypink	wildwatermelon	grey43
desertsand	pinegreen	wisteria	grey44
eggplant	pinkflamingo	yellow	grey45
electriclime	pinksherb	yellowgreen	grey46
fern	plum	yelloworange	grey47
forestgreen	purpleheart	grey01	grey48
fuchsia	purplemountainsmajesty	grey02	grey49
fuzzywuzzy	purplepizzazz	grey03	grey50
gold	radicalred	grey04	
goldenrod	rawsienna	grey05	
grannysmithapple	rawumber	grey06	
green	razzledazzlerose	grey07	

Figure 3.19: A list of the colours available in the WAVETRAIN plotter. This figure is a reproduction of the file colour_list.eps in the postscript_files directory. The US spelling “gray” may be used as an alternative to the UK spelling in any colour name: thus “gray23” and “bluegray” are permitted. Note that the Crayola crayon colour names “gray” and “tan” have been replaced by “crayolagray” (or “crayolagrey”) and “crayolatan” respectively. In the first case this avoids any confusion with the grey scale colour names; in the second case it avoids confusion with the mathematical function tan.

wave period in parameter plane plots. These and other settings can all be changed by the user if desired, by editing `plot_defaults.input`, which contains detailed comments explaining the meaning of each setting. The settings are also explained in detail in §3.4. After editing `plot_defaults.input`, the plotter must be restarted in order for the changes to take effect.

3.3.3 Labelling of Contours of Constant Period

If `showperiodcontourlabels` is set to `yes` in `plot_defaults.input`, then the plot command `@period_contour` draws the specified contour(s) and annotates it/them with labels showing the value of the period. The exception to this is contours that have been designated as homoclinic (see §3.1.4), which are not labelled. By default, labels are placed where (and if) a contour crosses the edge of the plotting region. This is appropriate in some cases, but in others it does not give clear labels. Therefore the plot command `period_contour` provides the opportunity to override this default. When prompted, the user can enter one or more expressions of the form `<control_parameter_name> = <value>` or `<wave_speed_name> = <value>`. This causes the default labels to be omitted, and instead labels are placed wherever (and if) a contour crosses the specified values.

As an example, consider the plot shown in Figure 2.5 on page 21, which used default labelling for the two period contours. Instead, for the plot command

```
@period_contour 101
```

one could enter

```
c=0.8 c=0.5
```

when prompted about label locations. Similarly, for the plot command

```
@period_contour 102
```

one could enter

```
c=0.7 A=0.5 A=0.8 c=0.4
```

when prompted about label locations. (Note that pcodes 101 and 102 refer to the contours with periods 3000 and 80 respectively). The resulting plot is shown in Figure 3.20. Note that the label specification “`A=0.8`” for pcode 102 has no effect, since the contour does not cross this value.

3.3.4 Adding Additional Lines, Points and Text to Plots

The plot command

```
@line x1 y1 x2 y2
```

superimposes a line between (x_1, y_1) and (x_2, y_2) in the current plot. Similarly,

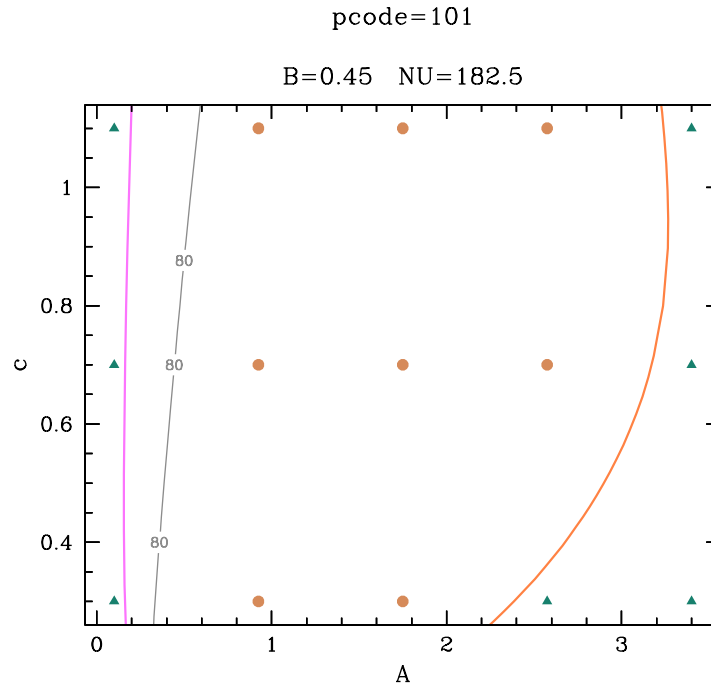


Figure 3.20: An illustration of manual specification of the label locations for contours of constant wave period. This figure is the same as Figure 2.5 except that in this case the default label locations for the contours have been overridden. The run and plot commands used to generate this figure are given in the main text, and are also listed in the Appendix.

`@point x y`

draws a point (symbol) at (x,y); the symbol type is specified in the `.style` file. For text labels, the command

`@text x y "my text"`

writes “my text” centred at the point (x,y). For all three commands, coordinates can be enclosed in double quotes. In `sm` this is optional, but in `gnuplot` it is essential if the coordinates are negative. Thus `@point -1.0 2.0` is invalid in `gnuplot` and gives an error; instead one must use `@point "-1.0" 2.0` or equivalently `@point "-1.0" "2.0"`.

As an example of the use of these commands, consider the bifurcation diagram plot in Figure 3.2b on page 79. This plot gives a detailed account of the existence of periodic travelling wave solutions as the wave speed c is varied for control parameter $A = 2.6$. However, it contains no information about stability, and indeed `WAVETRAIN` does not have the facility to compute stability in a bifurcation diagram. However, one can determine where the stability boundary curve crosses the line $A = 2.6$ via an optional argument to `stability_boundary`, for instance

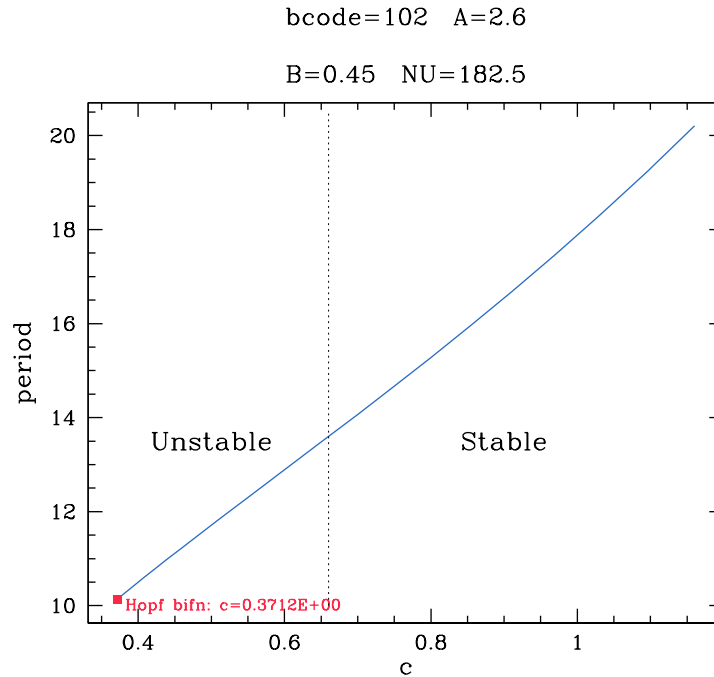


Figure 3.21: An illustration of the use of the plot commands `@line` and `@text`. These commands have been used to indicate the change in wave stability on a bifurcation diagram for the problem `demo`. The run and plot commands used to generate this figure are given in the main text, and are also listed in the Appendix.

```
stability_boundary demo 102 1.75 0.3 1.75 0.7 A=2.6
```

where 102 is an appropriate pcode value. This run reports that there is a change in wave stability at $A = 2.6$, $c = 0.66$. One can then add this information to the bifurcation diagram plot by typing

```
@line 0.66 10.0 0.66 20.5
```

at the plotter prompt, after drawing the bifurcation diagram. One might also wish to add labels indicating which part of the solution branch is stable via

```
@text 0.5 13.5 "Unstable"
```

and the command `@text 0.9 13.5 "Stable"` might be used in a similar way. The resulting plot is illustrated in Figure 3.21.

There are two other related plot commands: `@linesfile` and `@pointsfile`. These both plot data supplied by the user and contained in a file in the relevant input subdirectory; the file name is entered as a command argument. If `gnuplot` is being used as the plotter, the file name must be given in quotes (e.g. `"myfile.dat"`); in `sm`, this is optional.

The file must contain two columns of numerical data, which are a set of coordinates to be plotted; other columns are allowed in the file and will be ignored. Thus

```
@linesfile "myfile.dat"
```

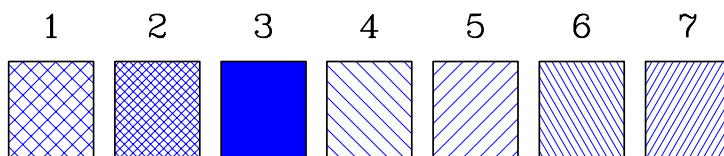
will draw lines connecting the various points whose coordinates are in the first two columns of `myfile.dat` in the relevant input subdirectory. The command `@pointsfile` has a second argument, specifying the point type to be used, according to the code numbers listed on page 142. Thus

```
@pointsfile "myfile.dat" 9
```

will draw a filled triangle at each of the points whose coordinates are in the first two columns of `myfile.dat` in the relevant input subdirectory. One potential use of the commands `@linesfile` and `@pointsfile` is to superimpose onto WAVETRAIN plots results from numerical simulations of the partial differential equations; an example of this is given in §2.3.25.

3.3.5 Shading Regions of the Control Parameter–Wave Speed Plane

When producing final plots for publication or presentation, it can be useful to shade parts of the control parameter–wave speed plane, for example to show regions in which there are multiple periodic travelling waves, or in which waves are stable or unstable. The plot command `@shade` is provided for this purpose. Its syntax is significantly more complicated than any of the other plot commands. There are either three or four arguments. The first two arguments specify the colour and pattern to be used for the shading; any WAVETRAIN colour is allowed (see §3.3.1), and the pattern is indicated by an integer between 1 and 7 as follows:



(If `gnuplot` is being used as the plotter, the relationship between the code number and the pattern is terminal dependent; the above applies for the default terminal `x11` and for postscript plots).

The third argument specifies the boundary of the region to be shaded. This boundary will be formed from parts of curves calculated by WAVETRAIN: Hopf bifurcation loci, period contours, fold loci or stability boundaries. Each of these parts is specified by a triplet of values, and the third argument consists of a series of these triplets, enclosed in double quotes ("). The first item in each triplet is a two letter code indicating the curve type: `hl`, `pc`, `fl`, `sb` denote Hopf bifurcation locus, period contour, fold locus or stability boundary respectively. The second item in the triplet is the `hcode` / `ccode` / `fcode` / `scode` number. The third item is one or more inequalities involving the control parameter or wave speed, to indicate which part of the curve is to be included. Multiple inequalities are separated by `&` (without any spaces), while if the whole curve is required then the inequality is replaced by `all`. Thus if the third argument to `@shade` is


```
"hl 102 k>2 pc 104 c<=4&k>1&c>1 sb 101 all"
```

then the shaded region will be that enclosed by the part of the Hopf bifurcation locus with hcode 102 for which $k > 2$, the part of the period contour with ccode 104 for which $1 < c \leq 4$ and $k > 1$, and the whole of the stability boundary with scode 101. (Here k and c are the control parameter and wave speed). The various curves forming the boundary of the region to be shaded must be specified in order, which can be either clockwise or anti-clockwise. If two consecutive curves do not have matching ends, then a straight line joining the ends will be inserted into the boundary of the shaded region; one important application of this is to regions bounded in part by the edge of the plot.

There is also an optional fourth command to `@shade`, which specifies an entry to be inserted into a key. This must be given in double quotes (`"`), and may contain multiple lines, with two back-slashes (`\\`) indicating a new line. The key is initialised by the `@ppplane` command. Then any subsequent run of `@shade` with a fourth argument adds an entry to the key. The key itself is generated by the additional plot command `@shade_key`. Note that the key is created as a separate plot; once postscript files of the (shaded) parameter plane and of the shading key have been generated, they can be combined within a word processor to give a composite figure if required.

As an example of using `@shade` and `@shade_key`, consider the plot shown in Figure 2.11. After issuing the various plotting commands required for this plot, one can shade the region corresponding to stable periodic travelling waves via the command

```
@shade red 1 "hl 101 c>0.91 pc 101 c>0.51 sb 101 all" "Stable
periodic\\travelling waves"
```

while the command

```
@shade blue 1 "hl 101 c<0.91 pc 101 c<0.51 sb 101 all" "Unstable
periodic\\travelling waves"
```

shades the region corresponding to unstable periodic travelling waves. (Note that these commands must be entered on a single line: the linebreak is only a necessity of the typesetting). The resulting plot is shown in Figure 3.22.

3.3.6 Record Keeping for Postscript Plots

When generating postscript plots for publication or presentation, users may wish to retain a record that would enable them to recreate the plot at a later date. This can be achieved by setting the plotter variable `keeprecords` to `yes` in `plot_defaults.input`. This setting causes the `@postscript` command to generate a plotting record as well as the postscript file. For example, the command `@postscript demoplot` creates the directory `demoplot_record` which contains two files. The first of these, `demoplot.tar`, is an archive of all of the data used for `demoplot.eps`; the second file, `demoplot.plt`, is a list of the various plot commands and option choices used to generate the plot, plus a listing of the plotter input files (see §3.4). Together these provide a full record that would enable the plot to be reproduced from scratch. This record-keeping facility should be used with care, simply because the `.tar` record files can be rather large. For this reason, the default setting of `keeprecords` is `no`.

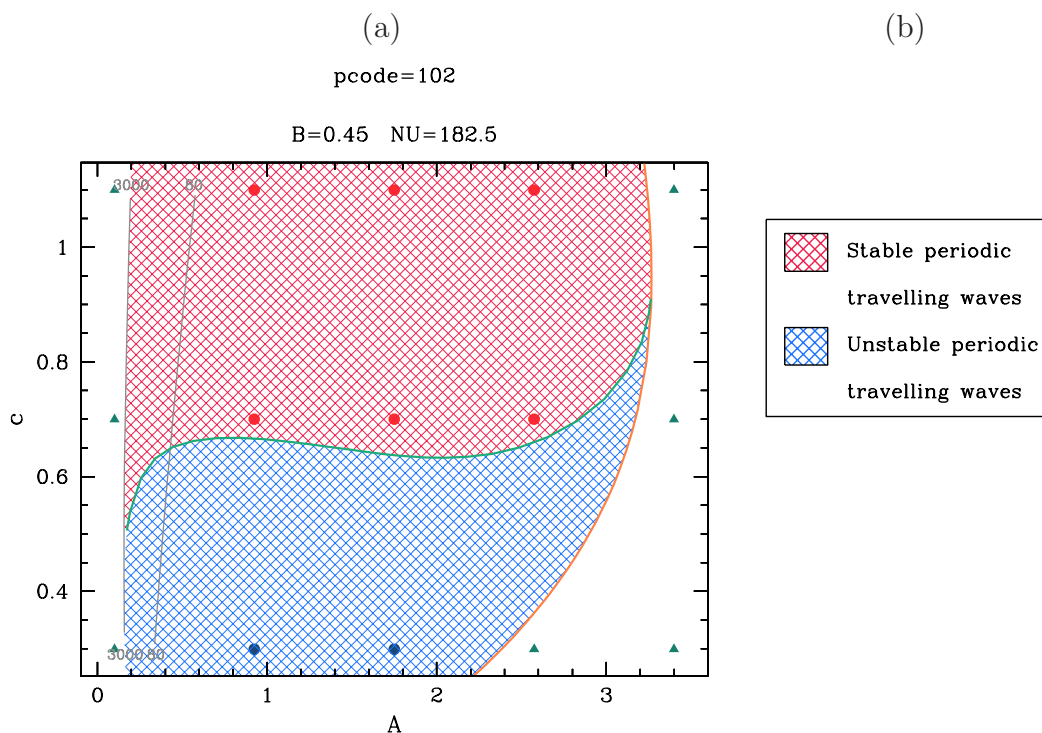


Figure 3.22: (a) An illustration of the use of the `@shade` command to shade regions in the control parameter–wave speed plane. This figure is the same as Figure 2.11 except that shading has been added to indicate the parameter regions giving stable and unstable periodic travelling waves. The run and plot commands used to generate this part of the figure are given in the main text, and are also listed in the Appendix. (b) The key to the shading used in part (a) of the figure. This key is generated by the `@shade_key` command.

3.3.7 The Bounding Box for Postscript Plots

The hard copies of plots produced by WAVETRAIN are encapsulated postscript files. Such files include the specification of a “bounding box”, that is, a rectangle completely surrounding the image. When using `gnuplot`, the user can expect correct bounding boxes in the (encapsulated) postscript files produced by WAVETRAIN’s `@postscript` (or `@ps`) command. However when using `sm`, the bounding box will usually be too big. This is an inevitable feature of the way in which postscript file geometry is controlled within `sm`. On many (but not all) unix-like operating systems, the tool `ps2eps` is available, and this is a simple way to correct an inappropriate bounding box. Therefore when using `sm`, the `@postscript` (or `@ps`) command asks users whether they would like to use `ps2eps` to correct the bounding box. WAVETRAIN does not check in advance whether the `ps2eps` tool is available on the user’s computer, and if it is not then an error will be reported (and no change will be made to the postscript file).

There are a number of other ways to correct bounding box sizes: typing “correct

bounding box” into a search engine gives a number of helpful links. Moreover, postscript files with bounding boxes that are too large can still be used effectively within documents. For example the `latex` command `includegraphics` permits the specification of a bounding box that overrides that given in the postscript file; on many systems appropriate bounding box coordinates can be found manually by viewing the file using `ghostview` and then moving the mouse to appropriate locations for the corners of the bounding rectangle.

3.3.8 Abbreviations for Plot Commands

Typically, users will want to enter plot commands repeatedly in relatively rapid succession, making the full command names rather cumbersome. Therefore, the plotter also recognises a number of two-letter shorthands:

```
@bd = @bifurcation_diagram
@fl = @fold_locus
@hl = @hopf_locus
@pc = @period_contour
@pp = @pplane
@ps = @postscript
@sb = @stability_boundary
@sp = @spectrum.
```

3.4 Details of Plotter Input Files

3.4.1 Details of `plot_defaults.input`

This file contains general default settings associated with plotting; the order of the various entries in the file does not matter. Settings concerning colours, line thicknesses and line styles are in the `.style` files, described in §3.4.2. The version of the file provided at installation contains detailed comments explaining the meaning of each setting.

3.4.1.1 Text expansion in `gnuplot`

The file `plot_defaults.input` contains a number of text expansion settings. If plotting is being done using `sm`, then the implementation of these settings is straightforward. If plotting is being done using `gnuplot`, then the text expansion settings will only take effect if enhanced text mode is specified in the terminal setting. (The terminal is set in this file, as part of the plotter setting “plotter”). In that case, the implementation of text expansion settings is straightforward for postscript plots, but it is somewhat complicated for screen plots. At installation, the terminal type specified for screen plots is `x11`, and it is anticipated that this will rarely be changed by users. The `x11` terminal type in `gnuplot` does allow changes in text size, but only in discrete jumps, not via a continuous expansion scale: allowable text sizes depend on the available font sizes for the selected font family. WAVETRAIN chooses the font size that is closest to the specified expansion factor. The run command `show_text_expansions` is provided as a tool to assist in the setting of text expansion factors in `plot_defaults.input` when plotting using `gnuplot`: it

lists the critical expansion factors at which text size changes. Note that these depend on the set of fonts that are defined on the user's computer. If a terminal type other than `x11` is specified for screen plots, then WAVETRAIN does not alter the text expansion settings given in `plot_defaults.input`, but users should be aware that if `gnuplot` cannot find a suitable font then it may substitute its internal default font type and size.

3.4.1.2 General default settings

`colourlistexpand` (setting at installation: 0.8)

This determines the font size for the list of colours created (in a postscript file) by the command `@set_colours`. Users plotting using `gnuplot` should read §3.4.1.1 before changing this setting.

`defaultstyle` (setting at installation: `colour`)

This determines the default style file that will be set when the plotter is started. The plotter style can be changed using the command `@set_style`.

`keeprecords` (setting at installation: `no`)

If this is set to `yes` then when a a postscript file is generated, a record is created of all of the run and plotter commands that were used to generated the plot. The record consists of two files, which have the names `file.tar` and `file.plt`, where `file.eps` is the name of the postscript file. These files are both placed in a subdirectory `file_record` of the directory containing `file.eps` (which is itself a subdirectory of `postscript_files`). One of these files, `file.tar`, is an archive of all of the output data files for the subdirectory and pcode value given in the `plot` command. Note that these output files contain a copy of all of the input data files used in the run commands. The other record file, `file.plt`, is a list of the plot commands used to create the plot, followed by a listing of `plot_defaults.input`, as it was when the plot command was given, followed by a listing of the style file (e.g. `colour.style`) that applied when the postscript file was generated. Taken together, these two record files contain all of the information that would be required to regenerate the plot from scratch. Setting `keeprecords` to `no` disables this record-keeping feature.

`labelsexpand` (setting at installation: 1.5)

This determines the font size for axes labels. Users plotting using `gnuplot` should read §3.4.1.1 before changing this setting.

`plotter` (setting at installation: `gnuplot x11 0 enhanced font "Helvetica,14"`)

This is the name of the plotter to be used, in lower case: either `sm` or `gnuplot`. For `sm`, the plotting device is taken from the user's `.sm` file. For `gnuplot`, the terminal type for screen plots must be specified after the word `gnuplot`, including options. In order for text expansion settings later in `plot_defaults.input` to take effect, these `gnuplot` terminal options must include the word `enhanced` (which specifies `gnuplot`'s enhanced text mode). Terminal options can also include the specification of a font, via

`font "<font_family_name>,<font_size>"`

The default font family is Helvetica and the default font size is 14. Other common font families include Courier, Lucida, Lucidabright, Lucidatypewriter, New Century Schoolbook, and Times. Note that the font family name can contain spaces and is not case sensitive: it is converted to lower case before searching for available fonts. Slant and weight options cannot be included; WAVETRAIN only uses roman, medium weight fonts. It should be emphasised that the font specification affects screen plots only: the font used for postscript plots is specified separately, via the variable `psfont`.

`psfont` (setting at installation: `Helvetica,14`)

This is only used for `gnuplot` (not `sm`): it is the font family name and base font size (separated by a comma) to be used for postscript plots. Note that this is specified independently from the font used for screen plots. Changing the base font size is the only way to change the size of tic mark labels (in `gnuplot`). It is recommended to use a font family name that is one of the 35 standard fonts defined on all postscript devices: lists of these can be found by typing “standard postscript fonts” into a search engine. Note that in contrast to the font specification for screen plots, the font family name for postscript plots is case sensitive. The plotter variable `psfont` was added in version 1.0 of WAVETRAIN, and to preserve back-compatibility its specification is optional, with the default being `Helvetica,14`.

`screenwidthchars` (setting at installation: 100)

This is only used for `gnuplot` (not `sm`), and then only when the terminal type setting is not `x11`. (For an `x11` terminal, the appropriate value is calculated automatically). It gives the width of the entire plotting canvas, as a number of average (unscaled) character widths. This will depend on both the `gnuplot` terminal type and the (unscaled) font type and size, which are specified in this file, as part of the plotter setting “`plotter`”. Note that the specified value can be either an integer or a floating point number.

`showtitles` (setting at installation: `yes`)

This determines whether or not titles (with parameter values etc) are shown above the plots.

`spacebottom` (setting at installation: 0.17)

`spaceleft` (setting at installation: 0.15)

`spaceright` (setting at installation: 0.02)

`spacetopnotitles` (setting at installation: 0.02)

`spacetoptitles` (setting at installation: 0.15)

These settings determine the amount of space around the graph(s), as a proportion of the overall plotting “canvas”. Thus the graph(s) itself occupies a region whose width is a fraction $(1 - \text{spaceleft} - \text{spaceright})$ of the canvas width, and whose height is a fraction $(1 - \text{spacebottom} - \text{spacetoptitles})$ of the canvas height if `showtitles` is set to `yes`, and a fraction $(1 - \text{spacebottom} - \text{spacetopnotitles})$ if `showtitles` is set to `no`. The space around the graph(s) contains tic mark labels, axes labels and (if selected) title(s). The size of this space does not scale with the text size selected

for axes labels and titles, and therefore may need to be adjusted if these are altered significantly from the defaults.

ticlabelsexpand (setting at installation: 1.3)

This is used for **sm** only, not **gnuplot**. It determines the font size for tic mark labels. This is included because it is convenient in **sm** to be able to change all text sizes relative to the default font, since the procedure for changing the default font in **sm** lies outside WAVETRAIN and would affect other uses of **sm**. Note that changing **ticlabelsexpand** also changes the size of the tic marks. For **gnuplot**, tic mark labels use the default font, which can be changed in the terminal specification, given as part of the plotter setting **plotter** elsewhere in this file.

titlesexpand (setting at installation: 1.5)

This determines the font size for titles. Users plotting using **gnuplot** should read §3.4.1.1 before changing this setting.

helpdisplay (setting at installation: 1)

This determines whether the output from the plotter command **@help** is displayed by scrolling (**helpdisplay=1**) or listing (**helpdisplay=2**) the help file.

3.4.1.3 Settings for plots of eigenvalue spectra

foldsymbolsizesp (setting at installation: 2.0)

This determines the size of the symbol used to indicate the location of the right-most non-zero turning point in the eigenvalue complex plane (if **showspectrumfold** is set to **yes**); this turning point is used to determine stability.

foldtextspexpand (setting at installation: 0.98)

This determines the size of the font used for the real part of the eigenvalue at the right-most non-zero turning point in the eigenvalue complex plane (if **show-spectrumfold** is set to **yes**); this turning point is used in the determination of stability, and also as a starting point for the calculation of stability boundaries of Hopf type. Users plotting using **gnuplot** should read §3.4.1.1 before changing this setting.

periodiceigenvaluessize (setting at installation: 1.2)

This determines the size of the symbols used to denote periodic eigenvalues (if **showperiodiceigenvalues** is set to **yes**) on spectrum plots.

restrictphasedifference (setting at installation: **no**)

This determines whether all or only parts of the spectrum is plotted, corresponding to a particular range of values of the phase difference across one period of the wave. Normally this should be set to **no**, but occasionally **yes** is useful for diagnostic purposes. In that case, the plotter command **spectrum** asks for keyboard entry of the desired range of phase difference values.

showperiodiceigenvalues (setting at installation: **yes**)

For plots of eigenvalue spectra, this determines whether or not to label the approximations to the eigenvalues for periodic eigenfunctions, found via discretisation.

showspectrumfold (setting at installation: **yes**)

This determines whether or not there is a label showing the real part of the eigenvalue at the right-most non-zero turning point in the eigenvalue complex plane; this turning point is used to determine stability.

spectrumkeyheight (setting at installation: 6)

If **spectrumcolour** is set to **rainbow** or **greyscale** or **grayscale**, a key is included in spectrum plots showing the mapping from the colours to the phase difference across the periodic travelling wave. The height of this key is determined by **spectrumkeyheight**. It is the key height as a percentage of the overall height of the main plot, the key and the space between them. Any value less than or equal to zero causes the key to be omitted.

spectrumkeyspacing (setting at installation: 3)

If **spectrumcolour** is set to **rainbow** or **greyscale** or **grayscale**, a key is included in spectrum plots showing the mapping from the colours to the phase difference across the periodic travelling wave. In **sm**, **spectrumkeyspacing** specifies the spacing between this key and the main plot, as a percentage of the overall height of the main plot, the key and the space between them. In **gnuplot**, the role of **spectrumkeyspacing** is the same, but it does not have such a precise meaning: larger values give more space between the key and the main plot, but precise automated control is not possible since the size of the main plot is somewhat unpredictable. Note that in **gnuplot**, manual rescaling of the main plot may be needed to achieve the required size; the user is prompted for keyboard input concerning this rescaling. Note also that in **gnuplot**, a negative value of **spectrumkeyspacing** can be used to reduce the gap between the key and the main plot: there may be a gap even if **spectrumkeyspacing** is set to zero. A negative value is never appropriate in **sm**.

spectrumkeytextexpand (setting at installation: 1.7)

This determines the font size used in the key. Users plotting using **gnuplot** should read §3.4.1.1 before changing this setting.

3.4.1.4 Settings for control parameter – wave speed plots

periodcontourlabelsdps (setting at installation: 0)

This specifies the number of decimal places used when labelling period contours. This applies whether **periodcontourlabelsstyle** is set to **e** or to **f**.

periodcontourlabelsexpand (setting at installation: 0.9)

This determines the font size for the labels indicating the value of the period along contours of constant period for periodic travelling wave solutions. Note that if this is changed, then **xperiodcontourlabelbox** and **yperiodcontourlabelbox** will usually need to be changed also.

periodcontourlabelsstyle (setting at installation: **f**)

This takes the value **e** or **f**, meaning exponential or floating point respectively.

For example, with `periodcontourlabelsdps` set to 3, a period of 34.5621 will appear as 34.562 if `periodcontourlablesstyle` is set to `f`, and as 0.346e+2 if `periodcontourlablesstyle` is set to `e`.

`pplanecodesexpand` (setting at installation: 0.9)

This determines the font size of the text used to denote run outcomes if `pplanetype` is set to `rcodes` (or equivalents), or to `period` (or `periods`), or if these settings are selected manually.

`pplanegap` (setting at installation: 0.06)

The axes limits in control parameter – wave speed plots are set wider than the limits in `parameter_range.input`, to allow for the size of the `rcodes`/periods/symbols used to indicate the solution outcomes. The amounts added to the axes ranges are a fraction `pplanegap` of the ranges in `plot_range.data` if this file is present in the output subdirectory for the specified `pcode` value, and in `parameter_range.input` otherwise. (The file `plot_range.data` will be present if either of the commands `add_points_list` or `add_points_loop` have been run). Note however that if the parameter plane was generated using the `new_pcode` command, then the limits in `parameter_range.input` are used and `pplanegap` is not used.

`pplanekeygaplength` (setting at installation: 2)

This determines the length of the gaps between line segments and text in the key, as a percentage of the overall width of the key. It also determines the spacing between the shading sample and the text in shading keys generated by `@shade_key`. The value must be an integer.

`pplanekeylinelength` (setting at installation: 8)

This determines the length of the line segments in the key, as a percentage of the overall width of the key. It also determines the size of the box used for shading samples in shading keys generated by `@shade_key`. The value must be an integer.

`pplanekeylineseparation` (setting at installation: 7)

This determines the spacing of the lines in the key, and in shading keys generated by `@shade_key`. Note that this spacing is not scaled by the font size selected for the text in the key, which is determined by `pplanekeytextexpand`, which is also defined in this file. Therefore if `pplanekeytextexpand` is changed, it may also be necessary to change the value of `pplanekeylineseparation`, to avoid the text on different lines overlapping.

`pplanekeylist` (setting at installation: "{ 1 5 2 3 4 11 12 13 14 15 16 }")

Note the inverted commas and curly brackets, which are both essential. This setting specifies which dots and line segments are used in the key for the control parameter - wave speed parameter plane plots. It also determines the order in which the dots (corresponding to outcome codes) and line segments are placed in the key. Numbers 1-5 indicate dots corresponding to the outcome codes, whose meaning is as follows:

- 1: Periodic travelling wave found, stability not requested. This is the outcome code when `ptw.loop` is run and a periodic travelling wave is found.
- 2: Periodic travelling wave found, and the spectrum shows that it is stable.
- 3: Periodic travelling wave found, and the spectrum shows that it is unstable.
- 4: No periodic travelling wave was found, with continuation along the periodic travelling wave solution branch ending for some reason other than a convergence problem.
- 5: No periodic travelling wave was found, with continuation along the periodic travelling wave solution branch ending due to a convergence problem.

Two digit numbers indicate line segments as follows:

- 11: Hopf bifurcation loci.
- 12: Period contour.
- 13: Homoclinic bifurcation loci.
- 14: Stability boundary (Eckhaus type).
- 15: Stability boundary (Hopf type).
- 16: Fold loci.

Note that if `pplanekeytype` is set to 2, then 4 and 5 would produce the same entry in the key, and if the number 4 is included in the list, it is ignored.

`pplanekeysymbolssize` (setting at installation: 2.0)

This determines the size of dots in the key.

`pplanekeytextexpand` (setting at installation: 1.15)

This determines the font size used for text in the key, and in shading keys generated by `@shade_key`. Note that the spacing of lines in the key is determined by `pplanekeylineseparation`, which is also defined in this file. This is not scaled by `pplanekeytextexpand`, and therefore if `pplanekeytextexpand` is changed, it may also be necessary to change the value of `pplanekeylineseparation`, to avoid the text on different lines overlapping.

`pplanekeytype` (setting at installation: 1)

This takes the value 1 or 2, and affects both the key for control parameter–wave speed plots, and the plots themselves. If it is set to 1, then separate colours and symbol types are used for the cases of no periodic travelling wave being found with a convergence failure in the numerical continuation, and of no periodic travelling wave being found with convergence throughout the continuation. Correspondingly, separate entries (with explanatory text) appear in the key. If `pplanekeytype` is set to 2, then the same colour and symbol type are used for the two cases (specifically, the colour and symbol type allocated to the case of convergence failure). Correspondingly, only one entry appears in the key. It is anticipated that most users will set `pplanekeytype` to 1 during the research process, and to 2 when producing figures for presentation or publication.

pplanesymbolsize (setting at installation: 2.0)

This determines the size of the symbols used to denote run outcomes if **pplanetype** is set to **symbol** (or equivalents), or to **period** (or **periods**), or if these settings are selected manually.

pplanetype (setting at installation: **rcode**)

Allowable settings are:

dot/dots/symbol/symbols

code/codes/number/numbers/rcode/rcodes

period/periods.

(Note that the **sm** command **dot** is disabled via **overload** when **plot** is run). This determines whether parameter plane plots use symbols, **rcode** numbers or the period of the periodic travelling wave to denote the points in the parameter plane at which waves have been calculated. This is a default setting, and the user is given the opportunity to override it via keyboard input when the command **@pplane** is run. Typically symbols would be used for final (presentation/publication) plots. Using **rcode** numbers is helpful for subsequent plotting of wave forms or eigenvalue spectra; however it can make the plots very busy. Periods are helpful prior to calculating period contours. When the setting is **period/periods**, a symbol is used for parameter values for which there is no periodic travelling wave. Note also that periods below 10^{-9} are shown as zero, while those above 10^{10} are shown as **inf**.

showperiodcontourlabels (setting at installation: **yes**)

This determines whether contours of constant period are labelled with the period values.

xperiodcontourlabelbox (setting at installation: 0.04)

yperiodcontourlabelbox (setting at installation: 0.02)

When labels are used for a period contour, the contours are not plotted in a rectangular region centred on the label location. The dimensions of this region are $2 \times (\text{control parameter range}) \times \text{xperiodcontourlabelbox}$ and $2 \times (\text{wave speed range}) \times \text{yperiodcontourlabelbox}$. Note that these dimensions are unaffected by **periodcontourlabelsexpand** and therefore they must usually be changed if **periodcontourlabelsexpand** is changed.

3.4.1.5 Settings for bifurcation diagram plots

bdkeygaplength (setting at installation: 2)

This determines the length of the gaps between line segments and text in the key, as a percentage of the overall width of the key. The value must be an integer.

bdkeyheight (setting at installation: 6)

This determines the height of the key (if present). It is the key height *per line* of the key, as a percentage of the overall height of the main plot, the key, and the space between them. (Note that whether or not there is a key, and the number of lines in the key if there is one, both depend on which plotting options are selected).

bdkeylinelength (setting at installation: 8)

This determines the length of the line, as a percentage of the overall width of the key. The value must be an integer.

bdkeyspacing (setting at installation: 4)

This determines the spacing between the key and the main plot, as a percentage of the overall height of the main plot, the key and the space between them.

bdkeytextexpand (setting at installation: 0.9)

This determines the font size used in the key.

foldsymbolsizedbd (setting at installation: 2.0)

This determines the size of the symbols used to indicate the Hopf bifurcation point and also, if **showbdlabls** is set to **yes** or **symbols**, folds and end points.

foldtextbdexpand (setting at installation: 0.98)

This determines the size of the font used to label the Hopf bifurcation point, folds and endpoints, if **showbdlabls** is set to **yes**.

showbdlabls (setting at installation: **yes**)

Allowable settings are: **yes/no/symbols**. This determines whether folds and end points are marked by symbols (if **showbdlabls** is set to **yes** or **symbols**), and whether the value of the bifurcation parameter (either the control parameter or the wave speed) is written on the plot at Hopf bifurcation points, folds and end points (if **showbdlabls** is set to **yes**). The Hopf bifurcation point is always marked by a symbol, even if **showbdlabls** is set to **no**.

3.4.1.6 Settings for the @line, @point and @text commands

extrapointsized (setting at installation: 2.0)

This determines the size of symbols generated by the **@point** command.

extratextexpand (setting at installation: 1.6)

This determines the font size for text generated by the **@text** command. Note that this does not affect any other text (axes labels, titles etc).

3.4.2 Details of Plotter Style File

Plotter style files contain settings associated with the plot style (colours, line thicknesses and line styles); the order of the various entries in the file does not matter. At installation, two different style files are provided: **colour.style** and **greyscale.style**; both contain detailed comments explaining the meaning of the various settings. The following list gives the value of each setting in the installation version of **colour.style**, as an example of a typical value.

3.4.2.1 Colour Settings

The available colours are listed in the file `colour_list.eps` in the `postscript_files` directory. They are based on the standard list of Crayola crayon colours, supplemented by cyan, and by 50 shades of grey (with `grey01` being almost black and `grey50` being almost white). The US spelling “gray” may be used as an alternative to the UK spelling in any colour name: thus “gray23” and “bluegray” are permitted. Note that the Crayola crayon colour names “gray” and “tan” have been replaced by “crayolagray” (or “crayolagrey”) and “crayolatan” respectively. In the first case this avoids any confusion with the grey scale colour names; in the second case it avoids confusion with the mathematical function `tan`.

3.4.2.1.1 Colour settings for drawing periodic travelling wave solutions

`wavecolour` (setting in `colour.style`: `denim`)

This specifies the colour used for drawing wave solutions, via `@ptw`.

3.4.2.1.2 Colour settings for plots of eigenvalue spectra

`spectrumcolour` (setting in `colour.style`: `rainbow`)

This specifies the colour used for drawing eigenvalue spectra. Any defined colour is allowed, or `rainbow`, or `greyscale` or `grayscale`. These three settings cause the spectrum to be coloured according to the phase difference in the eigenfunction over one period of the periodic travelling wave, using either a rainbow or greyscale colour map. An additional permitted setting is `none`, in which case the spectrum is not drawn; this enables plotting of just the eigenvalues corresponding to periodic eigenfunctions. Important note: in `gnuplot`, if variable colouring is requested via `spectrumcolour="rainbow"` or “greyscale” or “grayscale”, the size of the main plot is somewhat unpredictable, and manual rescaling may be needed to achieve the required size; the user is prompted for keyboard input concerning this rescaling. Also, these settings cause ticmarks to be absent in `gnuplot`; ticmark labels are present, however. If desired, ticmarks could be added manually using the `@line` command.

3.4.2.1.3 Colour settings for control parameter – wave speed plots

`colour1` (setting in `colour.style`: `rawsienna`)

`colour2` (setting in `colour.style`: `orangered`)

`colour3` (setting in `colour.style`: `midnightblue`)

`colour4` (setting in `colour.style`: `vividviolet`)

`colour5` (setting in `colour.style`: `tropicalrainforest`)

These settings define the colours used in plots of the control parameter – wave speed plane, to indicate different outcomes. The same colour can be used for several outcomes, if desired.

- 1: Periodic travelling wave found, stability not requested. This is the outcome code when `ptw_loop` is run and a periodic travelling wave is found.

- 2: Periodic travelling wave found, and the spectrum shows that it is stable.
- 3: Periodic travelling wave found, and the spectrum shows that it is unstable.
- 4: If `pplanekeytype=1`, then this indicates that no periodic travelling wave was found, with continuation along the periodic travelling wave solution branch ending for some reason other than a convergence problem. If `pplanekeytype=2`, then this is not used.
- 5: If `pplanekeytype=1`, then this indicates that no periodic travelling wave was found, with continuation along the periodic travelling wave solution branch ending due to a convergence problem. If `pplanekeytype=2`, then this indicates simply that no periodic travelling wave was found.

`homocliniccolour` (setting in `colour.style: pinkflamingo`)

This specifies the colour used for the loci of homoclinic solutions (which are actually contours of waves of large period that have been designated as homoclinic).

`hopflocuscolour` (setting in `colour.style: mangotango`)

This specifies the colour used for the loci of Hopf bifurcations in the travelling wave equations.

`foldlocuscolour` (setting in `colour.style: eggplant`)

This specifies the colour used for the loci of folds in the periodic travelling wave solution branch.

`periodcontourcolour` (setting in `colour.style: grey28`)

This specifies the colour used for contours along which periodic travelling waves have constant period.

`sbdyeckhauscolour` (setting in `colour.style: green`)

This specifies the colour used for curve(s) indicating the boundary between stable and unstable waves, when the change of stability is of Eckhaus type.

`sbdyhopfcolour` (setting in `colour.style: violetblue`)

This specifies the colour used for curve(s) indicating the boundary between stable and unstable waves, when the change of stability is of Hopf type.

3.4.2.1.4 Colour settings for bifurcation diagram plots

`bdlabelscolor` (setting in `colour.style: scarlet`)

This specifies the colour used for text and symbols labelling folds, end points and the Hopf bifurcation point in bifurcation diagram plots.

`bdlinecolourmax` (setting in `colour.style: denim`)

`bdlinecolourmean` (setting in `colour.style: cerise`)

`bdlinecolourmin` (setting in `colour.style: denim`)

`bdlinecolournorm` (setting in `colour.style: junglegreen`)

`bdlinecolourperiod` (setting in `colour.style: denim`)

`bdlinecolourstst` (setting in `colour.style: turquoiseblue`)

These settings specify the colours used for drawing the various properties of the periodic travelling wave solution branch.

3.4.2.1.5 Colour settings for the @line, @point and @text commands

`extralinecolour` (setting in `colour.style`: black)

This specifies the colour of lines generated by the `@line` command. Note that this does not affect any other lines.

`extrapointcolour` (setting in `colour.style`: black)

This specifies the colour of points generated by the `@point` command. Note that this does not affect any other symbols.

`extratextcolour` (setting in `colour.style`: black)

This specifies the colour of text generated by the `@text` command. Note that this does not affect any other text (axes labels, titles etc).

3.4.2.2 Point Type Settings

The meaning of these settings is the same for `sm` and for `gnuplot` with `x11`, `wxt` or `postscript` terminals, except where noted below. However the meaning within `gnuplot` is terminal dependent, and if a different screen terminal is specified in `plot_defaults.input`, then the meanings may be different: this can be checked via `gnuplot`'s `test` command (see `gnuplot` documentation for details).

- 1: horizontal-vertical cross (like a + sign)
- 2: diagonal cross (like a “times” sign)
- 3: a star (like an asterisk sign)
- 4: open square
- 5: filled square
- 6: open circle
- 7: filled circle
- 8: open triangle
- 9: filled triangle
- 10: open upside-down triangle (like a “nabla” sign)
- 11: filled upside-down triangle (like a “nabla” sign)
- 12: open diamond (`gnuplot`)/filled hexagon (`sm`)
- 13: filled diamond (`gnuplot`)/filled hexagon (`sm`).

3.4.2.2.1 Point type settings for control parameter – wave speed plots

`pointtype1` (setting in `colour.style`: 7)

`pointtype2` (setting in `colour.style`: 7)

`pointtype3` (setting in `colour.style`: 7)

`pointtype4` (setting in `colour.style`: 5)

`pointtype5` (setting in `colour.style`: 9)

These five settings refer to the five possible run outcomes, as for the settings `colour1,...,colour5`.

3.4.2.2.2 Point type settings for the @line, @point and @text commands

`extrapointtype` (setting in `colour.style: 7`)

This setting is for symbols drawn with the `@point` command.

3.4.2.3 Line Thickness Settings

In these settings, larger numbers mean thicker lines

3.4.2.3.1 General line thickness settings

`linethickness` (setting in `colour.style: 3`)

This determines the thickness of all lines in all plots, except for those given special values below.

3.4.2.3.2 Line thickness settings for control parameter – wave speed plots

`homocliniclinethickness` (setting in `colour.style: 5`)

This determines the thickness of the loci of homoclinic solutions (which are actually contours of waves of large period that have been designated as homoclinic).

`hopflocuslinethickness` (setting in `colour.style: 5`)

This determines the thickness of the loci of Hopf bifurcations in the travelling wave equations.

`foldlocuslinethickness` (setting in `colour.style: 5`)

This determines the thickness of folds in the periodic travelling wave solution branch.

`periodcontourlinethickness` (setting in `colour.style: 3`)

This determines the thickness of contours of constant period.

`sbdyeckhauslinethickness` (setting in `colour.style: 5`)

This determines the thickness of curve(s) indicating the boundary between stable and unstable waves, when the change of stability is of Eckhaus type.

`sbdyhopflinethickness` (setting in `colour.style: 5`)

This determines the thickness of curve(s) indicating the boundary between stable and unstable waves, when the change of stability is of Hopf type.

3.4.2.3.3 Line thickness settings for bifurcation diagram plots

`bdlinethicknessmax` (setting in `colour.style: 3`)

`bdlinethicknessmean` (setting in `colour.style: 3`)

`bdlinethicknessmin` (setting in `colour.style: 3`)

`bdlinethicknessnorm` (setting in `colour.style: 3`)

`bdlinethicknessperiod` (setting in `colour.style: 3`)

`bdlinethicknessstst` (setting in `colour.style: 3`)

These settings determine the thickness of the curves showing the various properties of the periodic travelling wave solution branch.

3.4.2.3.4 Line thickness settings for the @line, @point and @text commands

`extralinethickness` (setting in `colour.style`: 1.5)

This specifies the thickness of lines generated by the `@line` command. Note that this does not affect any other lines.

3.4.2.4 Line Type Settings

These are specified by a code number, whose interpretation depends on whether the plotter being used is `sm` or `gnuplot`. For `gnuplot`, the number specifies the `gnuplot` line style of that number. This is terminal dependent: see `gnuplot` documentation for details. In particular the same number may give different line styles on the screen and in a postscript plot. This is clearly unsatisfactory but is an intrinsic feature of `gnuplot`. However two different line style are guaranteed to be the same for all terminal types including postscript files: -1 = solid and 0 = dotted. In fact these are the only line styles available for the `x11` terminal, which is the screen terminal set at installation. Therefore it is recommended that if `gnuplot` is being used as the plotter then only solid and dotted lines are used, with different line types further distinguished by colour or by different greyscale shades. Note that the `gnuplot` line style with a particular number also carries with it a colour, but this is ignored by `WAVETRAIN`, being overwritten by the appropriate colour setting (specified above). For `sm`: -1 = solid, 0 = dotted, 1 = short dashed, 2 = long dash, 3 = dot & short dash, 4 = dot & long dash, 5 = short dash & long dash. Curves/lines for which no setting is listed below are always solid.

3.4.2.4.1 Line type settings for control parameter – wave speed plots

`homocliniclinetype` (setting in `colour.style`: -1)

This determines the line type of the loci of homoclinic solutions (which are actually contours of waves of large period that have been designated as homoclinic).

`hopflocuslinetype` (setting in `colour.style`: -1)

This determines the line type of the loci of Hopf bifurcations in the travelling wave equations.

`foldlocuslinetype` (setting in `colour.style`: -1)

This determines the line type of folds in the periodic travelling wave solution branch.

`periodcontourlinetype` (setting in `colour.style`: -1)

This determines the line type of contours of constant period.

`sbdyeckhauslinetype` (setting in `colour.style`: -1)

This determines the line type of curve(s) indicating the boundary between stable and unstable waves, when the change of stability is of Eckhaus type.

`sbdyhopflinetype` (setting in `colour.style`: -1)

This determines the line type of curve(s) indicating the boundary between stable and unstable waves, when the change of stability is of Hopf type.

3.4.2.4.2 Line type settings for bifurcation diagram plots

`bdlinetypemax` (setting in `colour.style`: -1)
`bdlinetypemean` (setting in `colour.style`: -1)
`bdlinetypemin` (setting in `colour.style`: -1)
`bdlinetypenorm` (setting in `colour.style`: -1)
`bdlinetypeperiod` (setting in `colour.style`: -1)
`bdlinetypest` (setting in `colour.style`: -1)

These settings determine the line types of curves showing the various properties of the periodic travelling wave solution branch.

3.4.2.4.3 Line type settings for the `@line`, `@point` and `@text` commands

`extralinetype` (setting in `colour.style`: 0)

This specifies the line type of lines generated by the `@line` command. Note that this does not affect any other lines.

Part 4

Reference Guide to WAVETRAIN

4.1 Alphabetical List of Run Commands

`add_points_list <subdirectory> <pcode>`

This command calculates the periodic travelling wave solutions and, if appropriate, their stability, for additional points in the control parameter–wave speed plane, following a previous run of `ptw_loop` or `stability_loop`. Stability will be calculated if the original run was of `stability_loop`, but not if the original run was of `ptw_loop`. The additional points are a list of control parameter and wave speed values, specified in the file `parameter_list.input` in the specified subdirectory of `input_files`. All other input files are copied from the record of those used in the original run of `ptw_loop` or `stability_loop`: the current files in the `input_files` subdirectory are not used, except for `parameter_list.input`, which must contain a list of the new points in the parameter plane. The required format is pairs of control parameter and wave speed values, separated by one or more spaces, with each pair on a separate line. Note that this command cannot be used for a pcode value generated using the `new_pcode` command. Note also that the additional points can lie outside the limits specified in `parameter_range.input` at the time of the initial run of `ptw_loop` or `stability_loop`; in that case the axes limits in subsequent control parameter–wave speed plots will be expanded to include the additional points.

`add_points_loop <subdirectory> <pcode>`

This command calculates the periodic travelling wave solutions and, if appropriate, their stability, for additional points in the control parameter–wave speed plane, following a previous run of `ptw_loop` or `stability_loop`. Stability will be calculated if the original run was of `stability_loop`, but not if the original run was of `ptw_loop`. The additional points are a grid of control parameter and wave speed values, as specified in the file `parameter_range.input` in the specified subdirectory of `input_files`. All other input files are copied from the record of those used in the original run of `ptw_loop` or `stability_loop`: the current files in the `input_files` directory are not used, except for `parameter_range.input`. Note that this command cannot be used for a pcode value generated using the `new_pcode` command. Note also that the additional points can lie outside the limits specified in `parameter_range.input` at the time of the initial run of `ptw_loop` or

`stability_loop`; in that case the the axes limits in subsequent control parameter-wave speed plots will be expanded to include the additional points.

`auto97test`

This command runs a test of the AUTO97 package, which is used extensively by WAVETRAIN. Typically, the command results in various compilation warnings. These warnings are compiler-dependent but typically do not cause any problems. However it is possible that the AUTO97 code will fail to compile even if the `fortrantest` command ran successfully, since AUTO97 departs somewhat from the Fortran77 standard. WAVETRAIN requires the user to have a Fortran77 compiler that can compile AUTO97. If compilation is successful, a test run will be performed, and the output from this will be compared with reference output.

```
bifurcation_diagram <subdirectory> <pname>=<pvalue>  
bifurcation_diagram <subdirectory> <speedname>=<speedvalue>  
with optional additional arguments variable=<name>  
and (repeated any number of times) file=<filename>
```

This command calculates the bifurcation diagram of the periodic travelling wave solutions as either the physical parameter or the wave speed are varied, with the other fixed at the value specified in the command line. The range of control parameter or wave speed values considered is that given in the file `parameter_range.input`. The first stage of the calculation is to look for Hopf bifurcations in the periodic travelling wave equations within this parameter range, and the command then calculates the solution branches emanating from each of these Hopf bifurcation points. Note in particular that the Hopf bifurcation limits given in `equations.input` are not used by this command. If one or more files are given in optional argument(s), then these are also used as starting points for calculating periodic travelling wave solution branches. If a variable name is given (one of the names listed in `variables.input`), then the properties of that variable (maximum, minimum, L2-norm, and average) are written to the output files, for use in plotting. Otherwise, plotting can only be done using the L2-norm of the whole periodic travelling wave solution, or the wave period. If optional arguments are used, the order of arguments two and above is arbitrary.

`cleanup`

This command deletes various temporary files created during calculations. It is run automatically as the final stage of commands if the constant `iclean` (defined in `constants.input`) is set to 1, but it can be run manually. Specifically, the command deletes all files in the directories `*/cleaned_up_files`, `*/data_files` and `temp_files`. None of these files are needed for plotting, but it is sometimes helpful to retain them after a run for debugging purposes.

```
convergence_table <subdirectory> <ecode> <evaluelist>
```

This command displays the results of a previous run of the `eigenvalue_convergence` command. The third and subsequent arguments are a list of eigenvalue numbers, between 1 and the total number of calculated eigenvalues. Alternatively `all` (or `All` or `ALL`) can be entered in place of `<evaluelist>` to give results for all of the

fortrantest

This command runs a test of the Fortran77 compiler, which is a system requirement. If the resulting screen output is “**Test completed**” then the compiler has passed the test. If an error message appears, such as “**f77: Command not found**”, this indicates that a compiler needs to be installed. A third possibility is that “**Test completed**” appears, but is preceded by a list of subroutines. This indicates that by default, the compiler operates in a non-silent mode. This type of compiler output will make the screen output produced by WAVETRAIN very difficult to follow, and it should be suppressed. This can be done by editing the file `input_files/defaults.input` and adding a suitable compiler flag on the line indicated. The appropriate flag is compiler dependent, but “**-silent**” or “**-fsilent**” are likely possibilities.

[illegible]

This command calculates a locus of Hopf bifurcation points in the travelling wave equations. The starting point for the locus is determined by searching for a Hopf bifurcation between the two specified points in the control parameter–wave speed plane. Only horizontal or vertical searches in this parameter plane are allowed, so that the arguments must satisfy either `pvalue1=pvalue2` or `speed1=speed2`.

```
list_crossings <subdirectory> <pcode> <scode>
```

```
list_crossings <subdirectory> <pcode> <cocode>
```

The commands `stability_boundary` and `period_contour` allow optional arguments specifying control parameter or wave speed values for which any crossing points of the period contour or stability boundary curve are to be recorded. This command lists these crossing points. If both `scode` and `ccode` output subdirectories exist for the code number given in the third argument, then the user is asked to specify to which the command refers. If the command reports that there are no crossing points, this may either be because no optional arguments were given to the `stability_boundary` or `period_contour` command, or because the boundary curve or contour did not pass through the specified values. Periodic travelling wave solutions corresponding to the crossing points are recorded in the relevant output subdirectory for period contours, but not for stability boundaries.

```
list_periods <subdirectory> <pcode>
```

This command lists the period associated with each period contour that has been calculated.

```
list_rcodes <subdirectory> optional:<pcode>
```

This command lists the control parameter and wave speed associated with each rcode value, plus the overall outcome of the calculation, and the period of the wave if one was found. If the value of pcode is omitted, it is taken to be 100, which corresponds to individual runs, rather than those performed via a loop through control parameter–wave speed values.

new_pcode <subdirectory>

This command creates a new output directory and copies the input files into this new directory. It does no calculations. The command is intended to enable Hopf bifurcation loci, period contours, and stability boundaries to be calculated without running `ptw_loop` or `stability_loop`.

pathtest

This command examines the search path to check whether it includes the current directory. If not, it attempts to determine which unix shell is being used, and on the basis of this produces screen output advising the user on how to change the search path. Note that because the search path might not include the current directory when this command is run, it should be entered as `./pathtest`.

period_contour <subdirectory> <pcode> <pvalue> <speed> <optional arguments>
period_contour <subdirectory> <pcode> period=<period> <pvalue1>
 <speed1> <pvalue2> <speed2> <optional arguments>

This command calculates a contour of fixed period for periodic travelling wave solutions. In the first usage, the wave is located for the given values <pvalue> and <speed>, and the contour is drawn for the period of this wave. In the second usage, the contour is drawn for the specified value of the period. In this case, the periodic travelling wave with <pvalue1> and <speed1> is found, and then the starting point for the contour is determined by searching the control parameter-wave speed plane between (<pvalue1>,<speed1>) and (<pvalue2>,<speed2>), looking for a wave with the required period. Only horizontal or vertical searches in this parameter plane are allowed, so that the arguments must satisfy either `pvalue1=pvalue2` or `speed1=speed2`. Note that it is necessary that a periodic travelling wave exists for (<pvalue>,<speed>) in the first usage, and for (<pvalue1>,<speed1>) in the second usage. However, it is not necessary for there to be a wave for (<pvalue2>,<speed2>) in the second usage. In both usages, the computational constant `iwave` is used (see §3.1.9 for details), and it is required that the periodic travelling wave branch does not have a fold between (<pvalue1>,<speed1>) and (<pvalue2>,<speed2>): no check is made on this during execution. The optional arguments have one of two possible forms: `<control_parameter_name>=<value>` or `<wave_speed_name>=<value>`, and WAVETRAIN records any points at which the contour crosses the specified values, and the corresponding periodic travelling wave solutions. These crossing points can be listed using the `list_crossings` command; they are not used in plotting. Data files containing the corresponding periodic travelling wave solutions are in the output subdirectory for the relevant pcode and ccode (see §4.4.4).

plot <subdirectory> optional:<pcode>

This command runs the plotter. If the value of pcode is omitted, it is taken to be 100, which corresponds to individual runs, rather than those performed via a loop through the control parameter-wave speed plane.

ptw <subdirectory> <pvalue> <speed>

This command calculates the periodic travelling wave solution for the specified values of the control parameter and wave speed. Stability of the wave is not calculated.

`ptw_loop <subdirectory>`

This command calculates the periodic travelling wave solutions for a grid of points in the control parameter–wave speed plane, as specified in `parameter_range.input`. Stability of the waves is not calculated.

`rmrcode <subdirectory> <bcode>`

`rmrcode <subdirectory> all`

This command deletes all output files associated with the bifurcation diagram calculation with the specified bcode value, and also deletes the corresponding output directory. If the second argument is `all` (`all` and `ALL` are also allowed), then the files and directories associated with all bcode values are deleted.

`rmccode <subdirectory> <pcode> <ccode>`

`rmccode <subdirectory> <pcode> all`

This command deletes all output files associated with the period contour calculation with the specified ccode value, and also deletes the corresponding output directory. If the second argument is `all` (`all` and `ALL` are also allowed), then the files and directories associated with all ccode values are deleted.

`rmecode <subdirectory> <ecode>`

`rmecode <subdirectory> all`

This command deletes all output files associated with the eigenvalue convergence calculation with the specified ecode value, and also deletes the corresponding output directory. If the second argument is `all` (`all` and `ALL` are also allowed), then the files and directories associated with all ecode values are deleted.

`rmfcode <subdirectory> <pcode> <fcode>`

`rmfcode <subdirectory> <pcode> all`

This command deletes all output files associated with the fold locus calculation with the specified fcode value, and also deletes the corresponding output directory. If the second argument is `all` (`all` and `ALL` are also allowed), then the files and directories associated with all fcode values are deleted.

`rmhcode <subdirectory> <pcode> <hcode>`

`rmhcode <subdirectory> <pcode> all`

This command deletes all output files associated with the Hopf bifurcation locus calculation with the specified hcode value, and also deletes the corresponding output directory. If the second argument is `all` (`all` and `ALL` are also allowed), then the files and directories associated with all hcode values are deleted.

`rmrcode <subdirectory> <pcode>`

`rmrcode <subdirectory> all`

This command deletes all output files associated with all calculations for the specified pcode value, and also deletes the corresponding output directory. If the second

argument is `all` (`all` and `ALL` are also allowed), then the files and subdirectories associated with all pcode values are deleted, followed by the (empty) output directory for `<subdirectory>`. Various prompts are issued to confirm that the user wishes to proceed.

```
rmmps <subdirectory> <name>
```

```
rmmps <subdirectory> all
```

This command deletes the postscript file `<name>.eps` for the specified subdirectory, and also deletes the corresponding plotting record if there is one. If the second argument is `all` / `All` / `ALL`, then the command first checks whether the file `all.eps` / `All.eps` / `ALL.eps` exists in the specified postscript files subdirectory. If so, the file and any associated record are deleted; if not, all files and associated records in that subdirectory are deleted.

```
rmrcode <subdirectory> <rcode>
```

```
rmrcode <subdirectory> <pcode> <rcode>
```

This command deletes all output files associated with the periodic travelling wave calculation with the specified rcode value, and also deletes the corresponding output directory. If no pcode value is specified then pcode is set to 100, corresponding to calculations that are run individually, rather than as part of a parameter plane loop. If pcode is greater than 100, the list of convergence failures is also updated.

```
rmrcode <subdirectory> <pcode> <rcode>
```

```
rmrcode <subdirectory> <pcode> all
```

This command deletes all output files associated with the stability boundary calculation with the specified scode value, and also deletes the corresponding output directory. If the second argument is `all` (`all` and `ALL` are also allowed), then the files and directories associated with all scode values are deleted.

```
set_homoclinic <subdirectory> <pcode> <ccode>
```

This command designates the period contour with the specified ccode value as being a homoclinic solution. There is no facility within WAVETRAIN to calculate the true loci of homoclinic solutions: rather these must be approximated by the loci of solutions with constant (large) period. This command should be used after a locus of large period has been calculated. It does not involve any new computation: it is just a change of labelling that effects the way in which the contour is plotted. Its effect can be reversed by the command `unset_homoclinic`. If the contour has already been designated as homoclinic then no error is reported, but the command has no effect.

```
set_worked_example_inputs <stage>
```

This command performs the various input stages of the worked example described in §2.3, up to and including the specified stage. Thus for example the command “`set_worked_example_inputs 5`” first empties the `workedx` input subdirectory (if it exists) and then performs the various steps in stages 1, 3 and then 5 of the worked example.

`show_text_expansions`

This command is provided as a tool to assist in the setting of text expansion factors in `plot_defaults.input` when plotting using `gnuplot`. For the standard `x11` terminal setting, text expansion in screen plots in `gnuplot` is not continuous; rather, there is a discrete set of possible text sizes. This command lists the critical expansion factors at which the text size will change; these depend on the fonts defined on the user's computer. The expansion factors specified in `plot_defaults.input` will be replaced by the closest value in this list for screen plots. Note however that for postscript plots in `gnuplot`, text expansion occurs on a continuous scale, and will be based on the expansion factors given in `plot_defaults.input`. This command is not relevant to plotting using `sm`.

`stability <subdirectory> <pvalue> <speed>`

This command calculates the periodic travelling wave solution for the specified values of the control parameter and wave speed, and then calculates its stability.

`stability_boundary <subdirectory> <pcode> <pvalue1> <speed1> <pvalue2> <speed2> <optional arguments>`

This command calculates a curve separating regions of stable and unstable periodic travelling waves. The starting point for the curve is determined by searching for a change in stability between the two specified points in the control parameter-wave speed plane. Only horizontal or vertical searches in this parameter plane are allowed, so that the arguments must satisfy either `pvalue1=pvalue2` or `speed1=speed2`. The optional arguments have the form `<control_parameter_name>=<value>` or `<wave_speed_name>=<value>`; if present, the program records any points at which the stability boundary crosses the specified values. These crossing points can be listed using the `list_crossings` command; they are not used in plotting. Note that the computational constant `iwave` applies to calculations done using this command (see §3.1.9 for details), and it is required that the periodic travelling wave branch does not have a fold between (`<pvalue1>`,`<speed1>`) and (`<pvalue2>`,`<speed2>`): no check is made for this during execution. *Warning:* this calculation can be quite time-consuming. It may be helpful to experiment with the `stability` command in order to obtain starting points in the control parameter-wave speed plane that are relatively close together (and are on either side of the stability boundary). The trade-off in computation time between such experiments and a longer run of `stability_boundary` is dependent on the problem and on the various input parameters set in the file `constants.input`.

`stability_loop <subdirectory>`

This command calculates the periodic travelling wave solutions, and their stability, for a grid of points in the control parameter-wave speed plane, as specified in the file `parameter_range.input`.

`unset_homoclinic <subdirectory> <pcode> <cocode>`

This command designates the period contour with the specified `cocode` value as not being a homoclinic solution, reversing the effect of a previous use of the command

`set_homoclinic`. If the contour has not previously been designated as homoclinic then no error is reported, but the command has no effect.

`wt_help <command>`

This command gives a brief description of the syntax and use of the specified command, which can be either a run command or a plot command. If no argument is given, then the available run and plot commands are listed.

4.2 Alphabetical List of Plot Commands

`@bd <bcode>`

This is an abbreviation for `@bifurcation_diagram`.

`@bifurcation_diagram <bcode>`

These commands plot the bifurcation diagram associated with the specified bcode value. The user will be prompted to enter what should be plotted on the vertical axis (wave period, L2-norm of the solution etc). The default axes limits are based on the whole data file, but the user will be given an opportunity to override these limits; it is particularly useful to override the vertical axes limits when plotting the period close to parameter values for which there is a homoclinic solution. When the user selects to plot only one solution measure, the plot will only contain a single curve, making a key unnecessary, but in other cases the plot includes a key, showing the meaning of the various line types.

`@exit`

This command deletes some temporary files before exiting from the plotter. It is equivalent to the command `@quit`.

`@fl <fcode>`

`@fl all`

This is an abbreviation for `@fold_locus`.

`@fold_locus <fcode>`

`@fold_locus all`

These commands plot the fold locus associated with the specified fcode value, or with all fcode values if the argument is `all` (`All` and `ALL` are also allowed).

`@help optional:<command>`

This command provides a brief summary of the syntax and use of the specified command. If no argument is given then a list of plot commands is displayed.

`@hl <hcode>`

`@hl all`

This is an abbreviation for `@hopf_locus`.

`@hopf_locus <hcode>`

`@hopf_locus all`

These commands plot the Hopf bifurcation locus associated with the specified hcode value, or with all hcode values if the argument is `all` (`All` and `ALL` are also allowed).

@line <x_coordinate1> <y_coordinate1> <x_coordinate2> <y_coordinate2>

This command draws a line between the two specified points. When plotting with **gnuplot** (but not **sm**), negative coordinates must be enclosed in double quotes, e.g. "-1.2".

@linesfile <file_name>

This command plots a series of line segments connecting the points given in the first two columns of the specified file, which must be located in the relevant input subdirectory. If **gnuplot** is being used as the plotter, the file name must be given in quotes (e.g. "myfile.dat"); in **sm**, this is optional. The data file can contain more than two columns: other columns will be ignored.

@pc <cocode>

@pc all

This is an abbreviation for **@period_contour**.

@period_contour <cocode>

@period_contour all

These commands plot the period contour associated with the specified <cocode> value, or with all ccode values if the argument is **all** (**All** and **ALL** are also allowed).

@point <x_coordinate> <y_coordinate>

This command draws a single dot (or other symbol, as defined by **extrapointtype** in the plotter style file) at the specified location.

@pointsfile <file_name> <point_type>

This command plots a series of dots (or other symbols) at the location specified by the first two columns of the specified file, which must be located in the relevant input subdirectory. If **gnuplot** is being used as the plotter, the file name must be given in quotes (e.g. "myfile.dat"); in **sm**, this is optional. The second argument specifies the symbol type. The meanings of the symbol type code numbers are given on page 142 and also in the comments of the style file(s). The data file can contain more than two columns: other columns will be ignored.

@postscript <filename> optional:<aspect_ratio>

These commands cause subsequent plotter commands to be written to the file <filename>.eps in the appropriate subdirectory of **@postscript_files**. If the plotter setting **keeprecords** is set to **yes**, the command also causes the plotter command record file <filename>_record/<filename>.plt and the run command and data record archive <filename>_record/<filename>.tar to be generated, which contain full details of the plotter commands and output data (respectively) that were used to create the plot. The optional second argument specifies the aspect ratio of the plot (that is, the horizontal size divided by the vertical size); the default value is 1.0. Note that the aspect ratio applies to the entire plot including axes labels, titles, etc, rather than to the region within the coordinate axes. Note also that plot commands are actually written to a buffer, and are only output to the postscript file when either the plotter device is changed (usually with either the

`@screen` command, or with another `@ps` or `@postscript` command), or when the plotter is exited (via the `@exit` or `@quit` command).

`@pp`

This is an abbreviation for `@pplane`.

`@pplane`

This command plots the control parameter-wave speed plane, indicating the existence of periodic travelling waves, and their stability if that has been calculated. There are three different plot types, denoting the waves by their rcode numbers, their periods, or a symbol. The default is set in `plot_defaults.input`, and the user is asked whether to use this default or override it. A key to the colours and symbols used in this plot is given in the file `postscript_files/ppplane.key.eps`; the key also shows the colours and styles of curves that may be added by subsequent plot commands. This file is regenerated each time the plotter is started, and also whenever the `@set_style` command is used; the choice of which symbols/lines are included in the key is specified in `plot_defaults.input`.

`@pplane_key` optional:<width_scaling>

This command draws a key to the colours and symbols used in control parameter – wave speed plots. A postscript file with the key is regenerated each time the plotter is started, and also whenever the `set_style` command is used; the command `@pplane_key` can be used to display a version on the screen. The choice of which symbols/lines are included in the key is specified in the file `plot_defaults.input`. The optional argument is a scaling factor applied to the width of the box surrounding the key. WAVETRAIN attempts to choose a box width appropriate to the font size and other relevant settings, but this is by necessity a crude estimate, so that it may be useful to adjust it by providing an appropriate scaling factor.

`@ps` <filename>

This is an abbreviation for `@postscript`.

`@ptw` <rcode>

This command plots the periodic travelling wave solution, as a function of the travelling wave coordinate, for the specified value of rcode.

`@quit`

This command deletes some temporary files before exiting from the plotter. It is equivalent to the command `@exit`.

`@sb` <scode>

`@sb` all

This is an abbreviation for `@stability_boundary`.

`@screen`

This command sets the plotter to write to the screen, and clears the plotting area (i.e. it starts a new plot). This is typically used to begin a new screen plot after plotting to a postscript file. If the plotter being used is `gnuplot`, “the screen”

denotes whatever is set as the plotting terminal in `plot_defaultsinput`. If the plotter being used is `sm`, “the screen” denotes whatever is set as the default plotter device in the user’s `.sm` file.

@set_style <style_name>

This command sets the default colours, line thicknesses and line styles, by reading them from <style_name>.style (in the `input_files` directory). It then regenerates the postscript file containing the key to the lines and symbols used in plots of the control parameter–wave speed plane. For this reason, if a postscript file is currently being written then it is ended by the `@set_style` command, with plotting returned to the screen at the end of the command.

@shade <colour> <pattern> "<bounding_curves>" optional:"<key_text>"

This command shades a region of the control parameter–wave speed plane. Any WAVETRAIN colour can be used for shading (see Figure 3.19). The shading pattern is specified by an integer between 1 and 7 (see page 128). The (closed) boundary of the region to be shaded will be formed from parts of Hopf bifurcation loci, period contours, fold loci or stability boundaries. Each of these parts is specified by a triplet of values, and the third argument consists of a series of these triplets, enclosed in double quotes. The first item in each triplet is a two letter code indicating the curve type: `hl`, `pc`, `fl`, `sb` denote Hopf bifurcation locus, period contour, fold locus or stability boundary respectively. The second item in the triplet is the `hcode` / `ccode` / `fcode` / `scode` number. The third item is one or more inequalities involving the control parameter or wave speed, to indicate which part of the curve is to be included. Multiple inequalities are separated by `&` (without any spaces), while if the whole curve is required then the inequality is replaced by `all`. The various curves forming the boundary of the region to be shaded must be specified in order, which can be either clockwise or anti-clockwise. If two consecutive curves do not have matching ends, then a straight line joining the ends will be inserted into the boundary of the shaded region; one important application of this is to regions bounded in part by the edge of the plot. The key text is optional; if present, it specifies an entry to be inserted into the shading key. The text must be given in double quotes, and may contain multiple lines, with two back-slashes (`\\`) indicating a new line. The key is initialised by the `@pplane` command, and any subsequent run of `@shade` with a fourth argument adds an entry to the key. The key is plotted by the `@shade_key` command.

@shade_key optional:<width_scaling>

This command generates a key for the shading patterns used in previous runs of `@shade`. The key is initialised whenever `@pplane` is run. The key is created as a separate plot from the (shaded) parameter plane; once postscript files of both plots have been generated, they can be combined within a word processor to give a composite figure if required. The optional argument is a scaling factor applied to the width of the box surrounding the key. WAVETRAIN attempts to choose a box width appropriate to the font size and other relevant settings, but this is by necessity a crude estimate, so that it may be useful to adjust it by providing an

appropriate scaling factor.

@sp <rcode>

This is an abbreviation for **@spectrum**.

@spectrum <rcode>

This command plots the eigenvalue spectrum for the periodic travelling wave solution corresponding to the specified value of rcode; if stability was not calculated for that value of rcode, then an error is reported. If **spectrumcolour** (defined in the plotter style file) is set to “rainbow” or “greyscale” (or “grayscale”) then a key for interpreting the colours is included in the plot.

@stability_boundary <scode>

@stability_boundary all

This command plots the stability boundary associated with the specified scode value, or with all scode values if the argument is **all** (**All** and **ALL** are also allowed). The boundary can correspond to a change in stability of either Eckhaus or Hopf type, and this is indicated by the colour and/or style of the line.

@text <x_coordinate> <y_coordinate> <text_string>

This command writes a piece of text (which must be in double quotes if it contains spaces) centred at the specified point.

4.3 WAVETRAIN Directory Structure

4.3.1 The Structure of Subdirectories of the Main WAVETRAIN Directory

For the benefit of users wanting to delve into the inner workings of WAVETRAIN, this section summarises the overall directory structure of the WAVETRAIN code. The main WAVETRAIN directory contains a formal copyright and disclaimer statement in the file **COPYRIGHT_DISTRIBUTION_DISCLAIMER**, plus a file **README** containing the WAVETRAIN version name / number. All other files in this directory are relatively simple Bourne shell scripts for the various commands. Most of the WAVETRAIN code is divided into a series of subdirectories:

auto_code This subdirectory contains the required parts of AUTO97. Some very minor changes have been made to the AUTO code, mainly removal of the “unknown” file status, for consistency with other Fortran77 programs in WAVETRAIN. The command names have also been altered, so that for example the standard AUTO command **@r** has been replaced by **auto_r**. This prevents any interference with existing AUTO code for users who have previously installed AUTO. At the start of each run command, a check is made on the AUTO files **auto_code/include/auto.h** and **auto_code/include/fcon.h** to see whether the AUTO code needs to be recompiled: each run of WAVETRAIN is performed using the smallest possible values of the AUTO constants **NINTX**, **NDIMX**, **NBCX** and **NTSTX**, in order to reduce run times.

auto_test This subdirectory contains files associated with the command **auto97test**.

cleaning_up This subdirectory contains the shell scripts associated with the deletion of data files (see §3.1.11).

controller This subdirectory contains the basic shell scripts and a few associated files for all of the WAVETRAIN commands. These scripts control the overall structure of the commands; for example the **period_contour** command is controlled primarily by the file **periodcontour.script** which runs various programs contained in the **loci** subdirectory.

documentation This subdirectory contains a PDF file of this user guide. It also contains a sub-subdirectory **worked_example** containing files associated with the command **set_worked_example_inputs**.

gammazero This subdirectory contains files associated with the calculation of eigenvalues corresponding to periodic eigenfunctions.

help_files This subdirectory contains files associated with the help system for run commands. Note the plotter help files are stored separately, in **plotting/help_files**.

input_files This subdirectory was discussed extensively in §2.2. It contains the two files **defaults.input** and **plot_defaults.input**, plotter “style” files which have the extension **.style**, and a series of sub-subdirectories, each of which contains input files associated with a particular set of equations.

loci This subdirectory contains files associated with the calculation of the Hopf bifurcation loci, fold loci, contours of constant wave period, and stability boundaries.

managing_info This subdirectory contains all files associated with the writing of information and error messages to output files and the screen.

output_files This subdirectory will be discussed in detail in §4.4. It contains a series of sub-subdirectories, each of which contains output files associated with a particular set of equations. It also contains four files. The file **logfile** contains the code numbers of any runs performed with **info=1**. This setting is intended for background runs, and no output is written to the screen; the code numbers are needed in order to plot the results. Each successive entry to **logfile** is simply appended to the current file (or the file is created if it does not exist), and the file can be emptied or deleted by the user whenever convenient. The files **auto_compile.output** and **auto_fc_compile.output** contain any warning messages generated during the most recent compilations of the AUTO97 main source code and the AUTO97 file conversion code, respectively. If no warnings were generated then the files will exist but be empty. The fourth file in the **output_files** subdirectory is **README**, which gives the version of WAVETRAIN being used (e.g. **wavetrain1.0**).

plotting This subdirectory contains all programs associated with the WAVETRAIN plotter. General plotting programs are located in this subdirectory itself, while macros associated with a particular plotter (**gnuplot** or **sm**) are in sub-subdirectories. The

sub-subdirectory `help_files` contains the files associated with the plotter's help system. There is also a sub-subdirectory `temp_files`, which contains temporary data files generated during plotting; these are deleted by the `@quit` or `@exit` commands.

postscript_files This subdirectory contains the single file `colour_list.eps` (see §3.3.1), plus a series of subdirectories containing postscript files associated with runs of the `plot` command.

processing This subdirectory contains the shell scripts and text files that are used to process the files `variables.input` and `equations.input`. The scripts convert the file `variables.input` into a more usable form, and convert the file `equations.input` into Fortran77 code. There is also a sub-subdirectory `most_recent_run`, which contains details of the `equations.input` file and associated constants that were last converted to Fortran77. This is checked whenever a new run command is entered, to determine whether processing must be redone.

ptwcalc This subdirectory contains files associated with the calculation of periodic travelling wave existence and form, and also those associated with `bifurcation_diagram` calculations.

spectrum This subdirectory contains files associated with the calculation of the portions of the spectrum in between the eigenvalues corresponding to periodic eigenfunctions.

temp_files This subdirectory is a “dumping ground” for temporary files created during runs. The directory is emptied at the end of runs if `iclean` (set in `constants.input`) has the value 1. The files are not needed for plotting, but users wishing to retain them for debugging can do so by setting `iclean=2`.

The subdirectories `controller`, `gammazero`, `loci`, `ptwcalc` and `spectrum` all contain sub-subdirectories named `cleaned_up_files` and `data_files`, which are used as repositories for temporary files generated during runs. Roughly, the `<subdir>/cleaned_up_files` sub-subdirectories contain temporary program files, while the `<subdir>/data_files` sub-subdirectories contain temporary data files used mainly by programs in `<subdir>`, and `temp_files` contains temporary data files used in a variety of different subdirectories. Like `temp_files`, the `cleaned_up_files` and `data_files` sub-subdirectories are emptied at the end of each run if `iclean` (set in `constants.input`) has the value 1. The files are not needed for plotting, but users wishing to retain them for debugging can do so by setting `iclean=2` in `constants.input`.

4.4 Details of Output Data Files

In normal use, there is no need for the user to be concerned with the data files generated by WAVETRAIN: in particular, this is not required in order to visualise the results using the plotter. However, occasionally users may wish to export data from WAVETRAIN calculations into other software. To assist in this, this section summarises the structure of the WAVETRAIN `output_files` directory, and gives details of the data files that will be

present in the various output subdirectories after runs of WAVETRAIN commands. The information on output files applies to cases in which runs have been successful and periodic travelling waves have been detected: for instance, if a run of `ptw` does not find a periodic travelling wave, then clearly the data files associated with periodic travelling wave form will not be present.

4.4.1 The Directory Structure of WAVETRAIN Output

Output data falls into two basic categories: (i) data that is associated with control parameter-wave speed planes; (ii) data this is not associated with control parameter-wave speed planes. Output in category (i) is all associated with a particular pcode value (101–999), and it is stored in the directory `output_files/pcode<code_number>`. Within such a directory, there are a small number of data files, which are copies of the input files as they were when the directory was created. The remainder of the data is stored in a series of subdirectories:

`output_files/pcode<code_number>/ccode<code_number>`

contains data on calculations of the contour of constant wave period with the specified code number.

`output_files/pcode<code_number>/fcode<code_number>`

contains data on calculations of the fold locus associated with the specified code number.

`output_files/pcode<code_number>/hcode<code_number>`

contains data on calculations of the Hopf bifurcation locus associated with the specified code number.

`output_files/pcode<code_number>/rcode<code_number>`

contains data on calculations of periodic travelling wave existence, form and possibly stability. Note that rcode numbers are four digits (1001–9999) while all other code numbers in WAVETRAIN are three digits.

`output_files/pcode<code_number>/scode<code_number>`

contains data on calculations of the stability boundary with the specified code number.

Details of the files in these various subdirectories are given in the following subsections.

Data in category (ii) is all stored in the directory `output_files/pcode100`. Note that 100 is not a valid pcode value for a parameter plane run (which must have a pcode between 101 and 999); the directory name `pcode100` is simply chosen for compatibility. There are no files directly in the `pcode100` directory: all files are in subdirectories, which are of three possible types:

`output_files/pcode100/bcode<code_number>`

contains data on calculations associated with bifurcation diagrams with the specified code number.

`output_files/pcode100/rcode<code_number>`

contains data on eigenvalue convergence calculations associated with the specified code number.

`output_files/pcode100/rcode<code_number>`

contains data on calculations of periodic travelling wave existence, form and (possibly) stability associated with the specified (four digit) code number. These are runs done using either the command `ptw` or the command `stability`.

All of these various directories are created by WAVETRAIN as required. Thus if the first run of a WAVETRAIN command using the input directory `demo` (say) is `ptw`, then this command creates the directories `output_files/demo`, `output_files/demo/pcode100` and `output_files/demo/pcode100/rcode1001`; the next run of the same command will create just `output_files/demo/pcode100/rcode1002`.

Although all code numbers of a particular type are created sequentially, gaps can arise via the file deletion commands (see §3.1.11). For example, the first three runs of `bifurcation_diagram` for input directory `demo` will create the directories `output_file/-demo/pcode100/bcode1001`, `output_file/demo/pcode100/bcode1002` and `output_file/-demo/pcode100/bcode1003`. If the command

```
rmrcode demo 1002
```

is given, then the second of these three output directories will be deleted. A subsequent run of `bifurcation_diagram` will then “fill the gap” by re-creating the directory `output_file/demo/pcode100/bcode1002`.

4.4.2 pcode Subdirectories

As discussed in §4.3.1, all output data is contained in a subdirectory of a directory named `pcode<value>`, where “value” is in the range 100-999. The `pcode100` contains subdirectories with data files associated with runs of the commands `bifurcation_diagram`, `eigenvalue_convergence`, `ptw` and `stability`; there are no files in the `pcode100` directory itself. Directories `pcode101`–`pcode999` contain data associated with studies of control parameter–wave speed parameter planes. These directories themselves contain copies of the input files, and also the following files:

equations.f This is the Fortran77 file that is generated using `equations.input` and that forms the basis of all WAVETRAIN calculations. It is included simply as a reference for any users wishing to see the source code used by WAVETRAIN.

errors.txt If any errors or warnings occur during the run, they will be listed in this file; otherwise the file will be present but empty. Note that this file contains a compilation of all error and warning messages that are generated during all runs associated with the `pcode` value. The messages are duplicated in the `errors.txt` file in the relevant subdirectory (see below).

info.txt This file contains information on the run of whichever of **new_pcode**, **ptw_loop** or **stability_loop** created the subdirectory. This file itself contains only general information about the progression through the command: detailed information is contained in the separate **rcode** subdirectories. If the command **add_points_list** or **add_points_loop** has been run for this pcode value, then corresponding general information will be appended to this file.

onlyptw.data This file contains the character **n** or the character **y**, indicating (respectively) whether stability was or was not determined for runs with this pcode value. This file will not be present if the pcode directory was generated using the **new_pcode** command.

plot_range.data This two-line file will be present if the command **add_points_list** or **add_points_loop** has been run for this pcode. It contains the minimum (*column 1*) and maximum (*column 2*) values of the control parameter (*row 1*) and wave speed (*row 2*) in all of the **rcode** subdirectories. If **plot_range.data** is present, then it is used as the basis for axes limits in plots of the control parameter– wave speed plane. Otherwise the axes limits are based on the file **parameter_range.input**.

The computational output data itself is contained in a directory of one of seven types: **bcode<value>**, **ecode<value>** (which are subdirectories of **output_files/<subdir>/pcode100/**), **ccode<value>**, **fcode<value>**, **hcode<value>**, **scode<value>** (which are subdirectories of **output_files/<subdir>/pcode101/...**, **output_files/<subdir>/pcode999/**), and **rcode<value>** (which can be subdirectories of any pcode output directory). These seven subdirectory types all contain copies of the input files, and also all contain the following files:

command.txt This one-line file contains the WAVETRAIN command (with arguments) that generated the directory. This file is not present in **rcode<value>** subdirectories for pcode values 101-999, since these are not generated directly by a WAVETRAIN command; however it is present in **rcode<value>** subdirectories for pcode 100, and for all cases of the other five types of output subdirectory.

equations.f This is the Fortran77 file that is generated using **equations.input** and that forms the basis of all WAVETRAIN calculations. It is included simply as a reference for any users wishing to see the source code used by WAVETRAIN.

errors.txt If any errors or warnings occur during the run, they will be listed in this file; otherwise the file will be present but empty.

info.txt This file contains detailed information on the run. It is the key to understanding any unexpected results during the run; this file and **errors.txt** are the only output files that users are expected to want to access unless they are exporting data from WAVETRAIN. Note that the contents of **errors.txt** is replicated in **info.txt**.

The remainder of this section documents the other files that are present in the seven types of output directory.

4.4.3 bcode Subdirectories

These are associated with calculations of bifurcation diagrams. After a successful run of the `bifurcation_diagram` command, the following files will be present:

bd.data This is the main data file containing the calculated points along the period contour.

Column 1: the bifurcation parameter, which can be either the control parameter or the wave speed.

Column 2: the norm of the whole wave solution.

Column 3: the wave period.

Column 4: the maximum of the selected variable.

Column 5: the mean of the selected variable.

Column 6: the minimum of the selected variable.

Column 7: the norm of the selected variable.

Column 8: 1 or 0, according to whether this line is a genuine data point or a dummy line, indicating a break in the curve. Such dummy lines are used to separate different branches of periodic travelling wave solutions, associated with different Hopf bifurcations. If this column contains 0 then the data in the other columns is irrelevant.

Columns 4–7 are only relevant if the `bifurcation_diagram` command used the optional argument “`variable=`”: otherwise they contain the specified information for the first travelling wave variable listed in `variables.input`, but they are not used.

bd.stst This file contains the calculated value of the steady state from which periodic travelling waves bifurcate.

Column 1: the bifurcation parameter, which can be either the control parameter or the wave speed.

Column 2: the norm of the steady state.

Column 3: the steady state value of the selected variable.

Column 4: 1 or 0, according to whether this line is a genuine data point or a dummy line, indicating a break in the curve. Such dummy lines are used to delimit separate regions of steady state existence. If this column contains 0 then the data in the other columns is irrelevant.

Column 3 is only relevant if the `bifurcation_diagram` command used the optional argument “`variable=`”: otherwise it contains the specified information for the first travelling wave variable listed in `variables.input`, but this is not used. The file `bd.stst` is not present if a steady state was not found. This will of course have prevented the detection of Hopf bifurcation points, but the bifurcation diagram may still have been calculated successfully if one or more “`file=`” optional argument(s) were given to the command.

hb_and_folds.data This file contains information used to label Hopf bifurcation points, folds and end points on the bifurcation diagram.

Column 1: the bifurcation parameter, which can be either the control parameter or the wave speed.

Column 2: the norm of the whole wave solution.

Column 3: the wave period.

Column 4: the maximum of the selected variable.

Column 5: the mean of the selected variable.

Column 6: the minimum of the selected variable.

Column 7: the norm of the selected variable.

Columns 8 and above: a code number and explanatory text string indicating the label type: it is either “1 HOPF BIFN POINT”, “2 FOLD” or “3 END OF SOLN BRANCH”.

variable.data This file contains information about the specification of a plotting variable using the “**variable=**” option in the **bifurcation_diagram** command.

Line 1: the number of the selected variable (variables are ordered as listed in the **variables.input** file), or 1 if “**variable=**” was not used.

Line 2: the number of the selected variable (variables are ordered as listed in the **variables.input** file), or 0 if “**variable=**” was not used. This line indicates whether or not “**variable=**” was used.

Line 3: the name of the selected variable, or **L2norm** if “**variable=**” was not used.

Line 4: the name of the bifurcation parameter (which will either be the name of the control parameter or the name of the wave speed, as given in **variables.input**).

Line 5: the equality fixing whichever of the control parameter or wave speed is not the bifurcation parameter, as it appears in the command line.

If an optional argument **file=<filename>** is used with **bifurcation_command**, then the corresponding files **<filename>.soln** and **<filename>.params** will also be copied to this output subdirectory.

4.4.4 ccode Subdirectories

These are associated with calculations of contours of constant wave period. After a successful run of the **period_contour** command, the following files will be present:

outcome.data This file contains a code number (the “outcome code”) indicating the outcome of the calculation, followed by comments explaining the meanings of the various possible outcome codes.

periodcontour.contourdata This is the main data file containing the calculated points along the period contour.

Column 1: the control parameter.

Column 2: the wave speed.

Column 3: 1 or 0, according to whether this line is a genuine data point or a dummy line, indicating a break in the curve. Such a dummy line will always be present in the middle of the file, since the locus is calculated in two parts, corresponding to the two possible continuation directions at the calculated starting point. If this column contains 0 then the data in the other columns is irrelevant.

Column 4: always the two characters **hc**.

Column 5: 1 or 0, according to whether the period contour has or has not been designated as a homoclinic solution (see §3.1.4).

Column 6: the period of the wave.

periodcontour.labels If labelling of period contours is selected as a plot option, then by default, labels are placed where (and if) the contours intersect the edge of the plotting region. This file contains the data used to plot these labels. Note that if the user overrides the default labelling when running the plot command **period_contour**, then this file will not be used. The contents of the file are:

Column 1: the control parameter.

Column 2: the wave speed.

Column 3: an orientation code specifying where the label is to be placed relative to the point specified in columns 1 and 2.

Column 4: the period of the wave.

Column 5: always the two characters **hc**.

Column 6: 1 or 0, according to whether the period contour has or has not been designated as a homoclinic solution (see §3.1.4).

period.data This one-line file contains the ccode value, the period of the contour, the outcome code, and then either “**hc 0**” or “**hc 1**”, indicating that the contour has not or has (respectively) been designated as homoclinic solution (see §3.1.4).

If optional arguments were given to the **period_contour** command (see §3.1.3), and if the period contour did cross the specified control parameter or wave speed value(s), then the following files will also be present:

pcsolution.list This file contains a list of the points at which the period contour crossed the values of the control parameter or wave speed specified in the optional arguments, together with the two-digit code number given to each crossing point. The contents of the file are:

Column 1: the code number, preceeded by “**Code=**”

Column 2: the control parameter, preceeded by “**<control_parameter_name>=**”

Column 3: the wave speed, preceeded by “**<wave_speed_name>=**”

Column 4: the period, preceded by “period=”. This will be the same for each row.

`pcsolution<code>.params` This one-line file contains the control parameter (first row) and wave speed (second row) for the crossing point with code number `<code>`.

`pcsolution<code>.soln` This file contains the periodic travelling wave solution for the crossing point with code number `<code>`.

Column 1: the travelling wave coordinate.

Columns 2 and above: the solution components of the periodic travelling wave, in the order that they are listed in `variables.input`.

The solution is given over exactly one period of the wave. Therefore the first column of the last row contains the period, and in columns 2 and above, the first and last rows are the same.

4.4.5 ecode Subdirectories

These are associated with eigenvalue convergence calculation. After a successful run of the `eigenvalue_convergence` command, the following files will be present:

`control_parameters.data` This two-line file lists the parameter values specified in the command line.

Row 1: the control parameter.

Row 2: the wave speed.

`convergence.table` This is the main data file used by the `convergence_table` command to generate convergence information.

Column 1: the value of `nmesh2`. The numbers in this column will be those given in arguments 4 and above in the command line.

Columns 2 and above: the real and imaginary parts of the eigenvalues corresponding to periodic eigenfunctions, and the estimated bound on the error in the eigenvalue. Thus the file has $(1+3\times\text{nevalues})$ columns in total. Columns 2 and 3 containing the real and imaginary parts of the eigenvalue with largest real part, and column 4 contains the estimated error bound for this eigenvalue. Columns 5 and 6 containing the real and imaginary parts of the eigenvalue with the next largest real part, and column 7 contains the estimated error bound for this eigenvalue, etc. Note that output will include either all eigenvalues or just those with non-negative real part, according to the value of `iposim` (which is set in `constants.input`). Note also that the error bounds are only approximate, and when one or more of the original partial differential equations do not contain time derivatives (i.e. (1.1b,c) applies) then the error bounds are particularly crude: see §3.1.13 for details.

`number_of_eigenvalues.data` This file contains only one number: $(100+\text{nevalues})$. Note that `nevalues` is set in `constants.input`.

4.4.6 fcode Subdirectories

These are associated with calculations of fold loci. After a successful run of the `fold_locus` command, the following files will be present:

`foldlocus.locusdata` This is the main data file containing the calculated points along the fold locus.

Column 1: the control parameter.

Column 2: the wave speed.

Column 3: 1 or 0, according to whether this line is a genuine data point or a dummy line, indicating a break in the curve. Such a dummy line will always be present in the middle of the file, since the locus is calculated in two parts, corresponding to the two possible continuation directions at the calculated starting point. If this column contains 0 then the data in the other columns is irrelevant.

Column 4: the period of the wave.

`outcome.data` This file contains a code number (the “outcome code”) indicating the outcome of the calculation, followed by comments explaining the meanings of the various possible outcome codes.

4.4.7 hcode Subdirectories

These are associated with calculations of Hopf bifurcation loci. After a successful run of the `hopf_locus` command, the following files will be present:

`hopflocus.locusdata` This is the main data file containing the calculated points along the Hopf bifurcation locus.

Column 1: the control parameter.

Column 2: the wave speed.

Column 3: 1 or 0, according to whether this line is a genuine data point or a dummy line, indicating a break in the curve. Such a dummy line will always be present in the middle of the file, since the locus is calculated in two parts, corresponding to the two possible continuation directions at the calculated starting point. If this column contains 0 then the data in the other columns is irrelevant.

Column 4: the period of the zero-amplitude wave. That is, 2π divided by the modulus of the imaginary part of the pair of purely imaginary eigenvalues.

`hopf.startdata` This one-line file contains the two pairs of control parameter and wave speed values specified in the command line.

`outcome.data` This file contains a code number (the “outcome code”) indicating the outcome of the calculation, followed by comments explaining the meanings of the various possible outcome codes.

4.4.8 rcode Subdirectories

These are associated with calculations of periodic travelling waves. They are generated by one of the four commands `ptw`, `stability`, `ptw_loop`, or `stability_loop`. In the first two cases, the parent directory will be `pcode100`, and in the third and fourth cases the parent directory will be `pcode101`–`pcode999`. If the subdirectory is generated using the `ptw` or `ptw_loop` command, with a periodic travelling wave being detected, then the following files will be present:

accurate_ptw.params This one line file contains parameters associated with the travelling wave solution.

Column 1: the control parameter.

Column 2: the wave speed.

Column 3: the period of the travelling wave solution.

Column 4: the number of points in the travelling wave solution. (This is the number of lines in the file `accurate_ptw.soln`).

accurate_ptw.soln This is the main data file containing the calculated periodic travelling wave solution. The solution that is recorded is the one calculated using `nmesh1`, which can be greater than `nmesh2` (these constants are set in `constants.input`).

Column 1: the travelling wave variable divided by the period. Therefore this column ranges from 0.0 to 1.0 in all cases.

Columns 2 and above: the solution components of the periodic travelling wave, in the order that they are listed in `variables.input`.

control_parameters.data This file contains key parameter values.

Row 1: the control parameter value.

Row 2: the wave speed value.

Row 3: the pcode value (100-999).

Row 4: the rcode value (1001-9999).

outcome.data This file contains a code number (the “outcome code”) indicating the outcome of the calculation, followed by comments explaining the meanings of the various possible outcome codes.

An rcode subdirectory generated using the `stability` or `stability_loop` command, with a periodic travelling wave being detected and with stability calculated successfully, contains the files listed above, and also the following additional files:

evalues.data This file contains the eigenvalues corresponding to periodic eigenfunctions.

Column 1: the real part of the eigenvalue.

Column 2: the imaginary part of the eigenvalue.

Column 3: a convergence code. If continuation along the spectrum starting from this eigenvalue is successful, the code is 0. Continuation along the spectrum is done in four stages: (1) continuation in a dummy parameter to obtain a suitable starting solution for the eigenfunction; (2) continuation for a few steps to establish the correct direction for continuation; (3) continuation backwards to obtain a starting point for which the phase shift in the eigenfunction is zero; (4) continuation to calculate the spectrum. If there is a convergence error during any of these stages, then the calculation terminates and the stage number in which the error occurred is written to this column. WAVETRAIN will then move on and begin continuation starting from the next eigenvalue corresponding to a periodic eigenfunction. Note that convergence errors typically occur because the discretisation used to calculate the eigenvalues corresponding to periodic eigenfunctions is too coarse, so that these eigenvalues are poor approximations to the true eigenvalues. Usually the remedy is to increase `nmesh2`, `nmesh3` or `order`.

Column 4: a code number for the eigenvalue, which ranges from 101 up to (100+`nvalues`).

Column 5: the estimated error bound for the eigenvalue. Note that the error bound is only approximate, and when one or more of the original partial differential equations do not contain time derivatives (i.e. (1.1b,c) applies) then the error bound is particularly crude: see §3.1.13 for details.

The eigenvalues are ordered in decreasing size of their real part.

failures.list This file contains a list of the eigenvalues corresponding to periodic eigenfunctions for which there was a convergence error in the spectrum continuation starting from that eigenvalue.

Column 1: the control parameter.

Column 2: the wave speed.

Column 3: the real part of the eigenvalue that was the starting point for continuation.

Column 4: the imaginary part of the eigenvalue that was the starting point for continuation.

Column 5: the continuation stage during which the convergence error occurred (=1, 2, 3 or 4) (see description of column 4 of `evalues.data`).

spectrum.max This one-line file contains data on the fold in the eigenvalue spectrum with largest real part, away from the origin.

Column 1: the real part of the eigenvalue at the fold.

Column 2: the imaginary part of the eigenvalue at the fold.

Column 3: the phase difference in the eigenfunction across one period of the wave, at the fold.

Column 4: the real part of the eigenvalue that was the starting point for continuation of the portion of the spectrum containing the fold. (This eigenvalue corresponds to a periodic eigenfunction).

Column 5: the imaginary part of the eigenvalue that was the starting point for continuation of the portion of the spectrum containing the fold. (This eigenvalue corresponds to a periodic eigenfunction).

Column 6: the code number of the eigenvalue that was the starting point for continuation of the portion of the spectrum containing the fold. The code numbers range from 101 up to (100+nevalues) (see description of column 4 of `evalues.data`).

If no folds are detected away from the origin in the calculated part of the spectrum, all columns contain zero.

spectrum.plotdata This is the main data file containing the calculated spectrum.

Column 1: the real part of the eigenvalue.

Column 2: the imaginary part of the eigenvalue.

Column 3: 1 or 0, according to whether this line is a genuine data point or a dummy line, indicating a break in the curve. If this column contains 0 then the data in the other columns is irrelevant.

Column 4: the phase difference in the eigenfunction across one period of the periodic travelling wave.

4.4.9 scode Subdirectories

These are associated with calculations of stability boundaries. After a successful run of the **stability_boundary** command, the following files will be present:

crossings.data Each line of this file is a sentence documenting a point at which the stability boundary curve crosses one of the control parameter or wave speed values specified via the optional arguments in the command line (see §3.1.2). If no optional arguments were given, or if optional arguments were given but no crossings were detected, then the file will be present but empty.

outcome.data This file contains either one or two code numbers (the “outcome code(s)”) indicating the outcome of the calculation, followed by comments explaining the meanings of the various possible outcome codes. If there is one code number, this refers to change of stability of Eckhaus type, and the number refers to the detection and tracing of the Eckhaus stability boundary curve. If there are two code numbers, this indicates that the search for a change of stability of Eckhaus type was completed without error but without finding an Eckhaus point, so that WAVETRAIN moved on to look for a change in stability of Hopf type. This is done first starting from the control parameter and wave speed values given in the third and fourth arguments, and then (if that is unsuccessful) starting from the control parameter and wave

speed values given in the fifth and sixth arguments. The two code numbers refer to the outcomes of these two calculations; the second code number is zero if the second calculation is not done, which will occur if the first calculation successfully detects and calculates a stability boundary curve.

stabilityboundary.boundarydata This is the main data file containing the calculated points along the stability boundary.

Column 1: the control parameter.

Column 2: the wave speed.

Column 3: 0, 1 or 2. The value 0 denotes a dummy line, indicating a break in the curve; in this case the data in the other columns is irrelevant. Such a dummy line will always be present in the middle of the file, since the locus is calculated in two parts, corresponding to the two possible continuation directions at the calculated starting point. The value 1 or 2 indicates that this is a genuine data point, with the change in stability being of Eckhaus or Hopf type respectively.

Column 4: the period of the wave.

Appendix: Commands Used to Generate Figures

This Appendix lists the various run and plot commands used to generate the graphical figures in this user guide. The code numbers assume that no previous runs have been performed, and that the commands are run in order. Therefore, for each command, it is assumed that all previous run commands listed in the table have been run. It is also assumed that after plotting each figure, the plotter is exited by typing either `@exit` or `@quit`. In all of these plots the default setting of `rainbow` is used for `spectrumcolour`. Consequently, if the plots are done using `gnuplot` rather than `sm`, there is an additional plotting option in which the user can alter the overall scaling of the plot. In all cases, the default scaling is appropriate.

Figure	Run commands	Plot commands	Plot options
2.2a	ptw demo 2.0 1.0 plot demo	@ptw 1001	
2.2b	ptw demo 1.5 0.8 plot demo	@ptw 1002	
2.3a	ptw_loop demo plot demo 101	@pplane	Plot type: rcode
2.3b	plot demo 101	@pplane	Plot type: period
2.5	hopf_locus demo 101 2.0 0.8 3.5 0.8 period_contour demo 101 period=3000 1.5 0.8 0.0 0.8 period_contour demo 101 period=80 2.5 0.8 0.5 0.8 plot demo 101	@pplane @hopf_locus 101 @period_contour 101 @period_contour 102	Plot type: symbol Label location: default Label location: default
2.7	stability demo 2.2 0.8 plot demo	@spectrum 1003	Axes limits: default
2.8	stability_loop demo plot demo 102	@pplane	Plot type: rcode
2.9a	plot demo 102	@spectrum 1002	Axes limits: default
2.9b	plot demo 102	@spectrum 1013	Axes limits: default
2.10	stability_boundary demo 102 1.75 0.3 1.75 0.7 plot demo 102	@pplane @stability_boundary 101	Plot type: rcode
2.11	copy_hopf_loci demo 101 102 copy_period_contours demo 101 102 plot demo 102	@pplane @hopf_locus 101 @period_contour all @stability_boundary 101	Plot type: symbol Label locations: default
2.12	set_worked_example_inputs 1 new_pcode workedex hopf_locus workedex 101 -1 0 -1 6 hopf_locus workedex 101 1 0 1 6 plot workedex 101	@pplane @hopf_locus 102	Plot type: rcode

Figure	Run commands	Plot commands	Plot options
2.13	<pre>set_worked_example_inputs 7 ptw_loop workedex hopf_locus workedex 102 1 0 1 6 plot workedex 102</pre>	<pre>@pplane @hopf_locus 101</pre>	Plot type: rcode
2.14	<pre>plot workedex 102</pre>	<pre>@ptw 1037</pre>	
2.17a	<pre>set_worked_example_inputs 14 stability_loop workedex plot workedex 103</pre>	<pre>@spectrum 1008</pre>	Axes limits: default
2.17b	<pre>plot workedex 103</pre>	<pre>@spectrum 1008</pre>	Axes limits: -0.036 0.002 -0.05 0.4
2.17c	<pre>set_worked_example_inputs 16 stability workedex -2.0 3.0 plot workedex</pre>	<pre>@spectrum 1022</pre> (The rcode number will be depend on how many test runs were performed in earlier stages)	Axes limits: default
2.18	<pre>plot workedex 103</pre>	<pre>@pplane</pre>	Plot type: rcode
2.19	<pre>stability_boundary workedex 103 0.857 2.171 0.285 2.171 stability_boundary workedex 103 -0.857 2.171 -1.428 2.171 plot workedex 103</pre>	<pre>@pplane @stability_boundary all</pre>	Plot type: symbol
2.20	<pre>set_worked_example_inputs 21 stability_boundary workedex 103 0.857 2.171 0.285 2.171 stability_boundary workedex 103 -0.857 2.171 -1.428 2.171</pre> <p>Note: the <code>stability_boundary</code> runs are not needed for this plot but are included here for consistency with the order in §2.3</p> <pre>plot workedex 103</pre>	<pre>@pplane</pre>	Plot type: period

Figure	Run commands	Plot commands	Plot options
2.21	<pre>period_contour workedex 103 period=50 -1.0 0.9 -0.2 0.9 period_contour workedex 103 period=100 0.85 1.6 0.85 2.2 period_contour workedex 103 period=150 0.9 2.2 0.2 2.2 period_contour workedex 103 period=200 1.0 2.6 -0.4 2.6 set_worked_example_inputs 24 hopf_locus workedex 103 1 0 1 6 plot workedex 103</pre>	<pre>@ppplane @hopf_locus 101 @stability_boundary 103 @stability_boundary 104 @period_contour all @pointsfile "pde_stable.data" 9 @pointsfile "pde_unstable.data" 8</pre>	<p>Plot type: symbol</p> <p>Label locations: delta=-1.8 delta=1.55</p>
2.22	<pre>new_pcode workedex copy_hopf_loci 103 104 copy_period_contours 103 104 copy_stability_boundaries 103 104 plot workedex 104</pre>	<pre>@ppplane @hopf_locus 101 @stability_boundary 103 @stability_boundary 104 @period_contour all @pointsfile "pde_stable.data" 9 @pointsfile "pde_unstable.data" 8</pre>	<p>Label locations: delta=-1.8 delta=1.55</p>
3.1	<pre>set_homoclinic demo 101 101 plot demo 101</pre>	<pre>@ppplane @hopf_locus 101 @period_contour all</pre>	<p>Plot type: symbol</p> <p>Label location: default</p>
3.2a	<pre>bifurcation_diagram demo c=0.8 plot demo</pre>	<pre>@bifurcation_diagram 101</pre>	<p>Vertical axis: norm</p> <p>Axes limits: both default</p>
3.2b	<pre>bifurcation_diagram demo A=2.6 plot demo</pre>	<pre>@bifurcation_diagram 102</pre>	<p>Vertical axis: period</p> <p>Axes limits: both default</p>

Figure	Run commands	Plot commands	Plot options
3.3a	bifurcation_diagram demo c=0.6 variable=W plot demo	@bifurcation_diagram 103	Vertical axis: norm max min Axes limits: both default
3.3b	plot demo	@bifurcation_diagram 103	Vertical axis: max mean stst Axes limits: both default
3.5a	ptw_loop demo1 plot demo1 101	@pplane	Plot type: symbol
3.5b	hopf_locus demo1 101 1.065 21.0 1.08 21.0 period_contour demo1 101 period=3000 1.085 20.0 1.08 20.0 set_homoclinic demo1 101 101 plot demo1 101	@pplane @hopf_locus 101 @period_contour 101	Plot type: symbol
3.6	bifurcation_diagram demo1 c=20.2 plot demo1	@bifurcation_diagram 101	Plot choice: norm Axes limits: both default
3.7a	Change grid to 50×15 in demo1/parameter_range.input ptw_loop demo1 plot demo1 102	@pplane	Plot type: symbol
3.7b	Change iwave to 2 in demo1/constants.input ptw_loop demo1 Reset iwave to 1 in demo1/constants.input Reset grid to 5×5 in demo1/parameter_range.input plot demo1 103	@pplane	Plot type: symbol
3.8	Change iwave to 0 in demo/constants.input ptw_loop demo copy_hopf_loci demo 101 103 (at warning: y) copy_period_contours demo 101 103 (at warning: y) Reset iwave to 1 in demo/constants.input plot demo 103	@pplane @hopf_locus 101 @period_contour all	Plot type: symbol Label location: default
3.9	fold_locus demo1 102 1.07 21 1.05 21 copy_hopf_loci demo1 101 102 copy_period_contours demo1 101 102 plot demo1 102	@pplane @hopf_locus 101 @period_contour 101	Plot type: symbol

Figure	Run commands	Plot commands	Plot options
3.16a	stability_loop demo3 plot demo3 101	@pplane	Plot type: rcode
3.16b	stability_boundary demo3 101 8.21E-4 9.75 8.22E-4 9.75 plot demo3 101	@pplane @stability_boundary 101	Plot type: symbol
3.17a	plot demo3 101	@spectrum 1003	Axes limits: default
3.17b	plot demo3 101	@spectrum 1002	Axes limits: default
3.20	plot demo 101	@pplane @hopf_locus 101 @period_contour 101 @period_contour 102	Plot type: symbol Label location: c=0.8 c=0.5 Label location: c=0.7 A=0.5 A=0.8 c=0.4
3.21	plot demo	@bifurcation_diagram 102 @line 0.66 10.0 0.66 20.5 @text 0.5 13.5 "Unstable" @text 0.9 13.5 "Stable"	Vertical axis: period Axes limits: both default
3.22a	plot demo 102	@pplane @hopf_locus 101 @period_contour all @stability_boundary 101 @shade red 1 "hl 101 c>0.91 pc 101 c>0.51 sb 101 all" "Stable periodic\\travelling waves" @shade blue 1 "hl 101 c<0.91 pc 101 c<0.51 sb 101 all" "Unstable periodic\\travelling waves" Note: Figure 3.22b must be plotted before exiting.	Plot type: rcode Label locations: default
3.22b		@shade_key	

References

E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, D. Sorensen: LAPACK Users' Guide (Third ed.), Society for Industrial and Applied Mathematics, Philadelphia, USA (1999).

F.V. Atkinson: Discrete and Continuous Boundary Problems, Academic Press, New York, USA (1964).

C. de Boor, B. Swartz: Collocation approximation to eigenvalues of an ordinary differential equation: the principle of the thing. *Math. Comp.* **35**, 679-694 (1980).

G. Boriougov, H. Engel: From trigger to phase waves and back again. *Physica D* **215**, 25-37 (2006).

F. Chatelin: The spectral approximation of linear operators with applications to the computation of eigenelements of differential and integral operators. *SIAM Rev.* **23**, 495-522 (1981).

E.J. Doedel, H.B. Keller, J.P. Kernévez: Numerical analysis and control of bifurcation problems: (I) Bifurcation in finite dimensions. *Int. J. Bifurcation Chaos* **1**, 493-520 (1991).

E.J. Doedel: AUTO, a program for the automatic bifurcation analysis of autonomous systems. *Cong. Numer.* **30**, 265-384 (1981).

J.J. Dongarra, J. Du Croz, S. Hammarling, R. J. Hanson: An extended set of FORTRAN basic linear algebra subprograms. *ACM Trans. Math. Soft.* **14**, 1-17 (1988a).

J.J. Dongarra, J. Du Croz, S. Hammarling, R.J. Hanson: Algorithm 656: an extended set of FORTRAN basic linear algebra subprograms. *ACM Trans. Math. Soft.* **14**, 18-32 (1988b).

J.J. Dongarra, J. Du Croz, I.S. Duff, S. Hammarling: A set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Soft.* **16**, 1-17 (1990a).

J.J. Dongarra, J. Du Croz, I.S. Duff, S. Hammarling: Algorithm 679: a set of level 3 basic linear algebra subprograms. *ACM Trans. Math. Soft.* **16**, 18-28 (1990b).

B. Fornberg: Calculation of weights in finite difference formulas. *SIAM Rev.* **40**, 685-691

(1998).

C.A. Klausmeier: Regular and irregular patterns in semiarid vegetation. *Science* **284**, 1826-1828 (1999).

H.O. Kreiss: Difference approximation for boundary and eigenvalue problems for ordinary differential equations. *Math. Comp.* **26**, 605-624 (1972).

C.L. Lawson, R.J. Hanson, D. Kincaid, F.T. Krogh: Basic linear algebra subprograms for FORTRAN usage. *ACM Trans. Math. Soft.* **5**, 308-323 (1979).

J.J. Moré, K.E. Hillstom, B.S. Garbow: The MINPACK project. In: Sources and Development of Mathematical Software, ed. W.J. Cowell, Prentice-Hall, Upper Saddle River, New Jersey, USA pp. 88-111 (1984).

J.D.M. Rademacher, B. Sandstede, A. Scheel: Computing absolute and essential spectra using continuation. *Physica D* **229**, 166-183 (2007).

J.D.M. Rademacher, A. Scheel: Instabilities of wave trains and Turing patterns in large domains. *Int. J. Bifur. Chaos* **17**, 2679-2691 (2007).

J.A. Sherratt: Pattern solutions of the Klausmeier model for banded vegetation in semi-arid environments II. Patterns with the largest possible propagation speeds. *Proc. R. Soc. Lond. A* **467**, 3272-3294 (2011).

J.A. Sherratt: Numerical continuation methods for studying periodic travelling wave (wavetrain) solutions of partial differential equations. *Appl. Math. Computation* **218**, 4684-4694 (2012).

J.A. Sherratt: History-dependent patterns of whole ecosystems. *Ecological Complexity* **14**, 8-20 (2013a)

J.A. Sherratt: Numerical continuation of boundaries in parameter space between stable and unstable periodic travelling wave (wavetrain) solutions of partial differential equations. *Adv. Comput. Math.* **39**, 175-192 (2013b).

J.A. Sherratt, M.J. Smith: Periodic travelling waves in cyclic populations: field studies and reaction-diffusion models. *J. R. Soc. Interface* **5**, 483-505 (2008).

B. Smith, J. Boyle, J. Dongarra, B. Garbow, Y. Ikebe, V. Klema, C. Moler: Matrix eigensystem routines, EISPACK guide. *Lecture Notes in Computer Science* **6**, Springer-Verlag, New York, USA (1976).

M.J. Smith, J.A. Sherratt: The effects of unequal diffusion coefficients on periodic travelling waves in oscillatory reaction-diffusion systems. *Physica D* **236**, 90-103 (2007).