Chapter 1

Installing AUTO.

1.1 Installation.

The AUTO file auto07p-0.9.1.tar.gz is available via `http://cmvl.cs.concordia.ca/auto`. Here it is assumed that you are using the Unix (e.g. *bash*) shell and that the file auto07p-0.9.1.tar.gz is in your main directory. See below for OS-specific notes.

While in your main directory, enter the commands `gunzip auto07p-0.9.1.tar.gz`, followed by `tar xvfo auto07p-0.9.1.tar`. This will result in a directory auto, with one subdirectory, auto/07p. Type `cd auto/07p` to change directory to auto/07p. Then type `./configure` , to check your system for required compilers and libraries. Once the configure script has finished you may then type `make` to compile AUTO and its ancillary software. The configure script is designed to detect the details of your system which AUTO requires to compile successfully. If either the configure script or the make command should fail, you may assist the configure script by giving it various command line options. Please type `./configure --help` for more details. Upon compilation, you may type `make clean` to remove unnecessary files.

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

The Graphical User Interface (GUI94) requires the X-Window system and Motif or LessTif. Note that the GUI is not strictly necessary, since AUTO can be run very effectively using the Unix Command Language User Interface (CLUI). Moreover, long or complicated sequences of AUTO calculations can be programmed using the alternative Python CLUI. The GUI is not compiled by default. To compile AUTO with the GUI, type `./configure --enable-gui` and then `make` in the directory auto/07p.

To use the Python CLUI and the "@pp" PyPLAUT plotter it is strongly recommended to install NumPy (`http://numpy.scipy.org`), TkInter, and Matplotlib (`http://matplotlib.sourceforge.net/`). Note that Matplotlib 0.99 or higher is recommended because it supports 3D plotting in addition to 2D plotting. Python itself needs to be at least version 2.3 or higher, but 2.4 or higher is strongly recommended and required for NumPy and Matplotlib. As of this writing Python 3.x is not yet supported by Matplotlib, and therefore not recommended. For enhanced interactive use of the Python CLUI it is also worth installing IPython (`http://ipython.scipy.org`).

The graphic tool for 3D AUTO data visualization, PLAUT04, is compiled by default, but depends on a few libraries that may not be in a standard installation of a typical Unix-like system. These libraries may be available as optional packages, though. In order of preference these are:

1. Coin3D (version 2.2 or higher), SoQt (1.1.0 or higher), and simage (1.6 or higher). With SoQt 1.5.0 or higher, simage is no longer required.

2. Coin3D with the SoXt library, which interfaces with (Open)Motif or LessTif (version 2.0 or higher) instead of Qt. The user interface has a few problems with LessTif though, in particular it is likely to crash on 64-bit machines, so the Qt version or (Open)Motif is recommended.

3. One can download SGI's implementation of the Open Inventor libraries from: `ftp://oss.sgi.com/projects/inventor/download/` Because SGI's implementation for Linux cannot show text correctly, we recommend that Coin is used instead of SGI's implementation.

The configure script checks for these libraries and outputs a warning if any of these libraries cannot be found. It first checks for SoQt, and then for SoXt, unless you pass `--disable-plaut04-qt` as an option to configure. If the libraries are not available you can still compile all other components of AUTO using make.

The Fortran code uses several routines that were not standardardized prior to the Fortran 2003 standard, for timing, flushing output, and accessing command-line arguments. The configure script first looks if the F2003 routines are supported (src/f2003.f90), then checks for a set of routines that are widely implemented across Unix compilers (src/unix.f90), and if that fails too, uses a set of dummy replacement routines (src/compat.f90), which could be edited for some obscure installations.

The PostScript conversion command `@ps` is compiled by default. Alternatively you can type `make` in the directory auto/07p/tek2ps. To generate the on-line manual, type `make` in auto/07p/doc, which depends on the presence of xfig's (transfig) fig2dev utility.

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

Some LAPACK routines used by AUTO for computing eigenvalues and Floquet multipliers are included in the package (Anderson, Bai, Bischof, Blackford, Demmel, Dongarra, Du Croz, Greenbaum, Hammarling, McKenney & Sorensen (1999)). The Python CLUI includes a slightly modified version of the Pointset and Point classes from PyDSTool by R. Clewley, M.D. LaMar, and E. Sherwood (`http://pydstool.sourceforge.net`)

1.1.1 Installation on Linux/Unix

A free Fortran 95 compiler, Gfortran, is shipped with most recent Linux distributions, or can be obtained at `http://gcc.gnu.org/wiki/Gfortran`. The following packages (and their dependencies) are recommended for Fedora:

- Python: python-matplotlib-tk and ipython.
- PLAUT04: SoQt-devel. For SoQt versions older than 1.5.0, to see pictures of stars, the earth and the moon instead of white blobs, compile simage from source (see `www.coin3d.org`; needs libjpeg-devel).
- PLAUT: xterm.
- GUI94: lesstif-devel or openmotif-devel.
- manual: LaTeX(tetex or texlive) and transfig.

and the following for Ubuntu and Debian:

- Python: python-matplotlib and ipython.
- PLAUT04: SoQt (libsoqt-dev or libsoqt4-dev) and libsimage-dev.
- PLAUT: xterm
- GUI94: lesstif2-dev or libmotif-dev.
- manual: LaTeX(tetex or texlive) and transfig.

Other distributions may have packages with similar names.

If you need to compile and install one of the above PLAUT04 libraries from the source code available at the above web site, and you find that, after that, PLAUT04 still does not work then you might need to adjust the environment variable `LD_LIBRARY_PATH` to include the location of these libraries, for instance `/usr/local/lib`.

1.1.2 Installation on Mac OS X

AUTO runs on Mac OS X using the above instructions provided that you have the development tools installed (see the Mac OS X Install DVD, Optional Installs, Xcode). You do not need to start an X server to run AUTO. Furthermore, the following packages are recommended:

- Gfortran: See `http://gcc.gnu.org/wiki/GFortranBinaries`. Alternatively, see `hpc.sourceforge.net` or `r.research.att.com/tools/`. At the time of writing the first of these is recommended to be able to benefit from parallelization.
- Python: you can install **32-bit** Python 2.7, NumPy, and Matplotlib .dmg files from `www.python.org`, `numpy.scipy.org`, and `matplotlib.sf.net`, respectively.
  For example, download from
  `http://www.python.org/ftp/python/2.7.2/python-2.7.2-macosx10.3.dmg`,
  `http://sourceforge.net/projects/numpy/files/NumPy/1.6.1/numpy-1.6.1-py2.7-python.org-macosx10.3.dmg/download`,
  and `http://sourceforge.net/projects/matplotlib/files/matplotlib/matplotlib-1.1.0/matplotlib-1.1.0-py2.7-python.org-macosx10.3.dmg/download`

The default system Python in Mac OS X is usable, but may be too old for Matplotlib and NumPy. There are other alternatives, for instance the Enthought Python Distribution at `www.enthought.com`. Fink should work but native graphics provided by the previous alternatives seem to work better.
To be able to plot in the Python CLUI in some versions of OS X, AUTO uses pythonw instead of python. This should happen automatically.

- PLAUT04: Get a Qt .dmg from `http://qt.nokia.com/downloads/qt-for-open-source-cpp-development-on-mac-os-x`. Similarly, you can get a binary Coin package from `coin3d.org`. After that you can compile SoQt (at least version 1.5.0) from the source code at `coin3d.org`. For example, download `http://ftp.coin3d.org/coin/bin/macosx/all/Coin-3.1.3-gcc4.dmg`, `http://get.qt.nokia.com/qt/source/qt-mac-opensource-4.8.0.dmg`, and `http://ftp.coin3d.org/coin/src/all/SoQt-1.5.0.tar.gz`.

  Try to make sure that the native (Aqua) Qt is used by setting `$QTDIR`, if you also have fink installed.

- PLAUT: In pre-Leopard OS X it appears that you do not see fonts. To solve this issue you need to obtain a different version of xterm; see `http://sourceforge.net/project/showfiles.php?group_id=21781`.

- GUI94: Perhaps possible using Fink but not attempted.

- manual: LaTeX and transfig (comes with xfig).

Notes for 64-bit Snow Leopard to be able to compile PLAUT04:

- To compile and install SoQt, after installing Qt and Coin3D, run

```
./configure CFLAGS="-m32" CXXFLAGS="-m32" LDFLAGS="-m32" FFLAGS="-m32"
make
sudo make install
```

- Then, in the AUTO-07p folder configure and compile AUTO as described above.

1.1.3 Installation on Windows

A native, light-weight solution for running AUTO on Windows is to use GFortran, MSYS 1.0.11 or higher (see `http://www.mingw.org`), combined with a native Win32 version of Python, obtained at `http://www.python.org`. To install this setup:

- Install Python (as of this writing, preferably version 2.7, not 3.2!) from `www.python.org`, NumPy from `numpy.scipy.org`, and Matplotlib from `matplotlib.sf.net`, which all come with installers.

- Install MinGW (make sure to include msys-base, gcc and fortran) using the MinGW Graphical Installer at `http://www.mingw.org`.

- Start MSYS using the Start menu (Start > All Programs → MinGW → MinGW Shell) or by clicking on its desktop icon, which puts you in a home directory, where you can unpack AUTO using `gunzip` and `tar`, as described above.
- Make sure that the `gfortran` and `python` binaries are in your `PATH`, and that their directories are at the front of it. You can do this, for instance, using the shell command `export PATH="/c/Python27:/bin:/c/Program Files/gfortran/bin:$PATH"` . You can also inspect, edit and then source the file `auto/07p/cmds/auto.env.sh` to achieve this.
- Now you should be able to run configure and make to compile AUTO as shown above.

You can use AUTO using shell commands from the default MSYS shell environment. You can also start the CLUI by double clicking on the file `auto.py` in the `python` folder of AUTO in Windows Explorer, or by creating a shortcut to it.

Alternatively, AUTO runs on Windows as above using the Unix-like environment Cygwin (see `http://www.cygwin.com`), but the non-Cygwin setup is more responsive and is much easier to setup for Matplotlib. You can however use its X server and lesstif to compile and run the old PLAUT and GUI94, if you so desire.

With some effort it is possible to compile PLAUT04 on Windows (without an X server) using Coin, SoQt, and Qt. You can also find precompiled PLAUT04 binaries at `http://sourceforge.net/projects/auto-07p/files`.

1.2 Restrictions on Problem Size.

There are no size restrictions in the file `auto/07p/include/auto.h` any more. This file now contains the default effective number of equation parameters `NPAR`, set to 36 upon installation. It can be overridden in constant files. See also Section 11.1. The default can be changed by editing `auto.h`. This must be followed by recompilation by typing `make` in the directory `auto/07p/src`.

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

1.3 Compatibility with Earlier Versions.

Unlike earlier versions, AUTO can no longer be compiled using a pure Fortran 77 compiler, but you need at least a Fortran 90 compiler. A free Fortran 95 compiler, GFortran, is shipped with most recent Linux distributions, or can be obtained at `http://gcc.gnu.org/wiki/GFortran`, which contains binaries for Linux, Mac OS X and Windows. AUTO was also tested with the free compiler `g95`, and there exist various commercial Fortran 9x compilers as well.

The AUTO input files are now called `c.xxx` (the constants file), and `h.xxx` (the HomCont constants file, only used with HomCont); the output files are called `b.xxx` (the bifurcation-diagram-file), `s.xxx` (the solution-file), and `d.xxx` (the diagnostics-file). The command `@rn` can be used to rename all these files from their old names. There are also minor changes in the formatting of these files compared to recent versions of AUTO, such as AUTO97 and AUTO2000.

The main change compared to AUTO97 is that there is now a programmable Python CLUI. The constants file can be written using a completely new, more flexible syntax, but the old syntax is still accepted, and files can be converted using the command `@cnvc` (see Section 5).

Due to the replacement of EISPACK routines by LAPACK routines for the computation of eigenvalues and eigenvectors, the sign of the eigenvectors may have flipped sometimes with respect to earlier versions. This affects the sign of some HomCont test functions and the initial direction when using the homotopy method (you may have to flip the sign of the starting distance in the routine STPNT). Eigenvectors are now normalized to avoid future problems and improve consistency.

Parameter derivatives in `DFDP` are now significant when using HomCont. If only the derivatives with respect to phase space variables are specified in `DFDU`, please use the setting `JAC=-1`.

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

A detailed list of user-visible changes can be found in the file `$AUTO_DIR/CHANGELOG`.

1.4 Parallel Version.

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

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

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

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

Both the Python CLUI and the commands in the `auto/07p/cmds` directory described in Chapter 5 may be used with the MPI version as well, by setting the `AUTO_COMMAND_PREFIX` environment variable. For example, to run AUTO in parallel using the MPI library on 4 processors just type `export AUTO_COMMAND_PREFIX='mpirun -np 4'` before you run the Python CLUI `auto` or the commands in `auto/07p/cmds` normally. The previous example assumed you are using the `sh` shell or the `bash` shell; for other shells you should modify the commands appropriately, for example `setenv AUTO_COMMAND_PREFIX 'mpirun -np 4'` for the `csh` and `tcsh` shells. Alternatively, inside the Python CLUI and scripts you can use `import os` followed by `os.environ["AUTO_COMMAND_PREFIX"]="mpirun -np 4"`.