DIII-D Equilibrium and Stability Modelling with Caltrans

R. H. Bulmer and L. D. Pearlstein

January 28, 2003 Revision 2

1 Introduction

The Caltrans executable (`caltrans`) consists of compiled and interpreted modules grouped into packages, where selected routines and variable identifiers may be referenced directly by the user. The Basis system provides the user interface to these public identifiers and includes a Fortran 90-like scripting language with which the user steers the code in a flexible way. This document is a brief guide to getting started modelling DIII-D equilibrium and stability problems with Caltrans.

DIII-D EFIT equilibria can be loaded into a Caltrans session from EQDSK files, then "morphed" into free-boundary and fixed-boundary (inverse) equilibria. Stability criteria can be evaluated with Caltrans routines, including a package that invokes Alan Glasser's DCON. Caltrans inverse equilibria can be easily modified (e.g., by altering the q-profile or the current profile). The modified equilibria can be saved in binary files for efficient loading into future sessions.

The usual mode of operation is interactive with the Basis parser interpreting and executing input on a line-by-line basis. When user input becomes lengthy it can be put into text files—scripts—and efficiently read into the code. In the extreme, the entire execution can be controlled by script files; thus operation in a batch-like mode is possible.

Section 2 covers the basics of using the code. Section 3 describes the experiment-specific loading of DIII-D EQDSK files and creation of a compatible Caltrans equilibrium. Section 4 describes the creation of free-boundary tokamak equilibria starting with parameters in a text file—the so-called *dead-start* procedure. Section 5 shows several ways to modify an equilibrium using the inverse solver (plasma profiles and boundary shape). Section 6 briefly describes some of the Caltrans-specific plot routines. Section 7 demonstrates saving equilibria to disk in portable binary files. Section 8 describes how to write EQDSK files (a-files, g-files and other formats). Section 9 discusses how to evaluate MHD stability criteria, including the use of the DCON package. Finally, section 10 introduces a script that demonstrates some of the concepts described in this document.

2 Session basics

Refer to the Caltrans web-page[1] for links to documentation—especially the Basis reference manual and its EZN graphics document—as Basis provides the user interface to Caltrans. New users are strongly encouraged to begin with the Basis tutorial.

Instructions for setting up your environment to execute the code are available through the "Unix Configuration for Using Basis/Corsica/Onetwo" link at the Caltrans web-page.

2.1 Session start-up

A session begins by invoking the Caltrans executable at the Unix shell prompt with optional command-line arguments. Typically the command-line contains at least two items: (1) a problem name string `-probname` *`pname`* specification, where *`pname`* is used as a prefix to name session output files[2], and (2) the name of a previously saved equilibrium, which must have a `.sav` suffix. Execute `caltrans -help` to display the full command-line documentation.

If one does not have a previously saved equilibrium, start-up with the DIII-D-specific morphing script file `d3.bas` by including its name on the command-line, then execute the `d3` procedure with a g-file name to create free- and fixed-boundary equilibria.

Start-up with a previously saved equilibrium:

```
caltrans -probname pname name.sav
```

or start-up from an EQDSK file:

```
caltrans -probname pname d3.bas
d3("gshot.time")
```

The `d3` procedure is described in more detail in §3. After start-up, the code will prompt for user input with a package-dependent prompt string.

2.2 Session termination

An interactive session ends when the user enters `quit` or `end`. If a batch job is being processed (see §2.3), be sure to include a termination command in your script. The function `quit` accepts an integer argument that is passed as the exit status, so you can say `quit(1)` to signal an error exit to a controlling process—of course the default exit status is zero.

[1] The Caltrans web-page URL is `http://web.gat.com/caltrans`.

[2] If a problem name prefix is not specified by the user, Caltrans will use the prefix of the first filename found on the command-line. If the code is launched with no command-line arguments, the default problem name string is `untitled`.

2.3 Batch-like operation

Caltrans is typically executed in an interactive mode but when repetitive or complicated commands are required it is best to put these into a text file using your favorite editor[3]. There are no restrictions on the file name, but `.bas` is a typical extension. The file can be read into the session with the Basis `read` statement or simply placed at the end of the command-line. Note the entire session can be executed in a batch-like way, with the last executing statement either `quit` or `quit(1)` (function call `quit(1)` can be invoked within error testing code-blocks as a robust way to abort a run).

Execute with file of commands in batch-like mode:

```
caltrans -probname pname your_script_file
```

A common mode of operation is to build up a script file by testing parts of it in an interactive session; when you are satisfied that the script does the intended task, use it in a batch-like way.

A simple Bourne shell script to process a Caltrans batch job that, say, creates a text file named `batch_job.dat`, looks like this:

```
#!/bin/sh
if caltrans batch_job.bas
then
   lpr batch_job.dat
fi
```

2.4 Session files

Files created automatically include the session log-file *pname*`.log` containing user–code dialog and NCAR graphics meta-files (see §2.5) *pname*`.`*nnn*`.ncgm`, where *nnn* is a sequence number. A graphics log file with a `.cgmlog` suffix will also be created for each `ncgm` file.

If Basis detects an error, then a *pname*`.`*nnn*`.err` file will be created. The error files may contain helpful information if the Basis debug-mode has been enabled (`debug=yes`), useful if you are developing a script of your own.

2.5 Viewing and post-processing graphics

Caltrans sends graphics output, by default, to the `ncgm` file. The user may additionally direct output to other files (see the Basis EZN document) or, more commonly, to an X-window for viewing during the session.

Open graphics windows with the Basis `win` command; to name the window

[3] Emacs users may be interested in the lisp code `basis.el` which aids the preparation of Basis and/or Caltrans scripts.

use the syntax: `win on name`. Caltrans also has an `ow(name)` macro. Graphics windows are closed with either `win close [name]` or `cw(name)`. Multiple graphics windows can be opened.

The NCAR command `idt` can be used after a session to view `ncgm` files, select specific frames for exportation, or for printing specific frames[4]. The `ctrans` command translates `ncgm` files to many other formats, including PostScript.

2.6 Built-in documentation

Basis has built-in documentation allowing the user to query the code. The top-level command is `help`, which introduces the `version`, `news` and `list` commands. The most useful is `list`—invoke it without an argument to get its own documentation, then invoke it with `list name` to display documentation about identifier *name*, e.g. `list probname`. All user accessible variables, functions (and subroutines) and macros respond to the `list` command. To get the *contents* of a variable, just enter its name.

The `list packages` statement is useful to get a list of the Caltrans packages. Packages are identified, for historical reasons, with 2–3 character names. Since it will occasionally be critically necessary to reference a specific package variable, it is important to be familiar with their package names (see §2.7).

2.7 Basis language features

The Basis language is Fortran 90-like and has the familiar constructs: `WHILE-ENDWHILE`, `IF-ELSEIF-ELSE-ENDIF`, `DO-ENDDO`, etc. Functions can be defined with `FUNCTION name(); <body>; ENDF` as well as macros. Multiple lines can be placed on the same physical line by separating them with semicolons, and lines can be continued by placing a backslash at the end of a line. The comment character is #.

A large set of Basis built-in mathematical routines are available, as well as many other useful tools for file I/O, string processing, matrix and array processing, etc. They are all documented in the Basis reference manual.

Variables can be created (and destroyed) on-the-fly. A common task is to set up a loop to do a parameter scan, execute some Caltrans routine within the loop (like an equilibrium solver) and capture results in user-created variable arrays. When the loop has finished, the results can be be saved to disk or graphed with the Basis plot routines. All Basis variables must be typed (`integer`, `real`, `character`, `logical` are some of the types available). There is also a `chameleon` type that, in an assignment statement, takes on the type of the r.h.s. Basis predefines the identifiers `$a`–`$z` as chameleon variables—they are commonly used in interactive sessions to avoid having to declare scratch variables.

[4]The man-page for `ncarv_spool` describes how to customize the post processing buttons for `idt`. See the `ncargintro` man-page for a complete list of NCAR graphics utilities.

Identifier name conflicts are resolved by qualification with the defining package name, using dot-notation. For example, the variable name `r` is popular, appearing in several packages as a user-accessible quantity. You can even make your own at the Caltrans prompt by issuing a command like `real r`. Running the `list` command on `r` will show all package occurrences, where user-created identifiers are assigned to package `global` (unless otherwise directed). The `list` command will show the packages in which an identifier appears and its *priority*. In case of conflicts, the identifier with the highest priority is inferred. To explicitly reference an identifier preface its name with its package name; for example, the equilibrium package definition of `r` is `eq.r`, the DCON definition is `dcn.r`, etc.

2.8 Reading script files

Script files are read into the code with the Basis `read` *`filename`* statement. User script files will typically be in the current working directory, but Basis has a search path of directories in which to search for files. The search path is customized by Caltrans to include directories where standard script files are kept, and directories where binary "generic" save-files are kept. The search path is stored in Basis variable `path`[5]; to see the search path, just enter `path`.

If you have Basis or Caltrans scripts of your own that you would like to be able to read into any session, put them in some particular directory and add the directory name to the Basis search path with a `pathadd("dirname")` statement. Statements such as a user's `pathadd` calls are good candidates to go into a personal Basis start-up file (`$HOME/.basis`) that will get read automatically each time Caltrans is launched. The user's current working directory is in the default search path.

Files interpreted with the `read` statement are processed line-by-line. Caltrans turns off line echoing, but for debugging user scripts it is often useful to turn on line echoing (to both the screen and the log-file) with an `echo=yes` statement. It can be turned off with `echo=no`, or sent to the log-file only with `echo=logonly`.

2.9 Reading and writing data

Basis has efficient facilities for reading and writing data in disk files. Text files are accessed with the stream I/O facility and binary files with the PFB (Portable Files from Basis) facility, which accesses binary data in self-describing files which are portable across all Unix platforms. These capabilities are briefly introduced by example in the following sub-sections, refer to the Basis reference manual for a complete description.

[5] Basis path-related variables and routines are part of the parser package (`par`) group `Path`, so do `list Path` to get a complete list. Note that Basis is *case-sensitive*: `Path` is the name of a group and `path` is the name of a variable.

2.9.1 Text file I/O

Small amounts of data can be imported into the code by simply including the information as part of a script, for example

```
real some_data=[1.2, 3.8, 7]
```

when embedded in a script file, will import the data when the script is read. For large amounts of data the stream I/O facility should be used.

Say you had 10000 measurements of some current as a function of time in text file `some_data.dat` arranged in two columns. First create storage for the data (in one big array); open the text file for read access; read in the data "all at once" with the stream input operator `>>`; close the file; then decompose the big array into a time array and a current array:

```
integer n=10000
real big_array(2,n)
integer io=basopen("some_data.dat","r")
io >> big_array
call basclose(io)
real time(n)=big_array(1,)
real current(n)=big_array(2,)
```

Exporting data to text files can be performed in various ways. One way is to simply redirect `STDOUT` to a file using the Basis `output` command:

```
output some_data.dat2
time; current
output tty # return STDOUT to screen
```

However, the format of the data in this case is determined by Basis.

The stream output operator `<<` and the `format` function can be used to tailor the output format. Let's say we want to write our data in 2-columns, with a header line and with the TAB character (9th ASCII character) separating data columns:

```
character*1 tab=char(9)
io=basopen("some_data.dat3","w")
io << " time current"
do $i=1,n
   io << format(time($i),10,3,1) << tab \
      << format(current($i),16,8,2)
enddo
call basclose(io)
```

In this example `time` is written in a Fortran-style `F10.3` format and `current` in an `E16.8` format. Note the use of chameleon variable `$i` as a temporary integer and the continuation character: \.

2.9.2 Binary file I/O

The PFB facility in Basis offers an efficient and portable way to handle large amounts of data. The underlying routines are part of the Portable Application Code Toolkit (PACT)[6].

In Caltrans sessions the PFB routines are frequently used to save (write) and restore (read) equilibria (see §7).

The user may also use the PFB routines in a customized way. As an example, following those in §2.9.1, let's say we were executing the Caltrans equilibrium solver many times to produce "snap-shots" representing the time-evolution of a discharge. At each step we *append*, in variables `time` and `current`, two particular quantites of interest. We might want to do things (e.g., make plots, write formatted files) with these data in some *future* session, but don't want to bother with those things now. Initially we would declare zero-length arrays to hold our data:

```
real time(0), current(0)
```

A loop might be used to execute the equilibrium solver and at the end of the loop we append to our storage arrays:

```
time := shotTime # shotTime is a caltrans variable
current := placur # code variable placur holds Itor
```

When the loop has finished, we write the data of interest into a binary file with statements like:

```
create some_data.pfb
write time, current
close
```

where the three commands `create`, `write` and `close` are Basis interface routines to the PFB library.

In some future session, we simply

```
restore some_data.pfb
```

and our arrays of `time` and `current` will be available.

If you have forgotten what you put in the PFB file, do:

```
open some_data.pfb
ls
```

to list its contents.

[6] PACT is a comprehensive system of portable software for scientific applications, see `http://pact.llnl.gov`.

2.10 Code interaction

Input to the code consists of Basis-language instructions, either coming a line at-a-time in an interactive session or read from a script file. The instructions contain combinations of routine (function or subroutine), macro and variable-name identifiers which are defined by Basis, Caltrans and by the user. These are woven together in the Basis-language constructs and parsed by the Basis parser. The parser employs the GNU `readline` facility, so Emacs-style input-line editing features—including searchable history—are available.

Interrupts are triggered with `CTRL-C` and place the code in *debug-mode*. In debug-mode you can query or alter variables or do any legitimate operation. Enter `cont` to resume an interrupted operation or enter `abort` (or any illegal token) to irreversibly interrupt the operation and return to the parser.

External processes can be executed with the `basisexe` routine, e.g.,

```
if (basisexe("ls foo") <> 0) then # no file "foo"
```

and when the command's exit status is of no interest, the bang syntax `!ls foo` can be used.

Basis functions may or may not have return values and the user may or may not "capture" return values. For example, the Basis `basisexe` function returns the exit status of the process it gave to the operating system. If invoked as a function:

```
basisexe("command") # or...
integer status=basisexe("command") # or...
if (basisexe("command") <> 0) then # error
```

the exit status will be, respectively: (a) echoed, (b) placed in new variable `status` or (c) used in an `IF` test. If called as a Fortran subroutine:

```
call basisexe("command")
```

the return-value (in this case the command's exit status) will be discarded.

3 Morphing EQDSK files with d3

The standard script-file `d3.bas` defines the `d3` procedure and a few auxiliary routines (described in §3.3). The `d3` procedure is used to "morph" an EFIT equilibrium into an equivalent Caltrans equilibrium for subsequent analyses (e.g., stability, transport, etc.). Morphed equilibria are often used as a starting point or baseline for developing modified equilibria (with varied shape, profiles, etc.).

The EFIT equilibrium is defined in `a` and `g` EQDSK files. It is recommended that both the g-file and a-file be available, although `d3` will attempt to do something sane with only the g-file information. To save time during the morphing

process, Caltrans should be initialized *without* a save-file on the command-line, allowing it to select an appropriate standard (generic) save-file with the same grid resolution as defined in the g-file, thereby expediting the morphing process. However, one may alternatively specify any DIII-D save-file on the command-line, e.g.,

```
caltrans any.sav d3.bas
```

The `d3` procedure creates a free-boundary equilibrium, saves it to disk in a portable binary save-file, then creates a fixed-boundary (inverse) equilibrium—using the free-boundary plasma shape—and saves the inverse-solve equilibrium in an inverse-save-file. These save-files can later be expeditiously restored into a Caltrans session without the need to re-execute `d3`.

During the morphing process graphics windows will automatically open showing comparisons of the Caltrans equilibrium results with the corresponding data from the g-file or a-file. After each plot is displayed, the code will pause for user confirmation. The user is encouraged to carefully compare the Caltrans equilibrium quantities with those from the EQDSK file and report[7] discrepancies.

3.1 Options to the d3 procedure

The full argument list for the `d3` procedure is

```
d3(g_file,confirm,fix_edge,make_inverse,msrf, \
thetac,inv_k,inv_p)
```

where all but the first argument are optional[8]. These arguments are described below.

g_file — names a g-file in the current Basis `path`. An a-file name will be constructed from the *g_file* argument by replacing the first character with an `a`. If the a-file and g-file are not in the current working directory, or in the Basis search path, then specify the *pathname* to the g-file (remember, the a-file must be in the same location as the g-file).

confirm — an integer that determines the type of action to be taken after each graphics frame is displayed. If *confirm* < 0, the code will pause and wait for a user response (default). If *confirm* $= 0$ no frames will be displayed. If *confirm* > 0 the code will pause for that number of seconds (to allow the user time to peruse the graphics), then continue automatically. When processing many g-files it may be more convenient to turn off the display; then the morphing process will proceed without user intervention. In this case, the user

[7] Problems of any nature can be e-mailed to `caltransbugs@alvin.llnl.gov`; please include a brief description of the problem and the location of any files needed to replicate it.

[8] Actually, all the arguments are optional—executing `d3` with no arguments will display a brief help message. Optional arguments take their default value unless otherwise specified. For example, to invoke the `d3` procedure only specifying the g-file name and `thetac=0.001`, execute: `d3(g-file,,,,,0.001)`.

should verify the results by viewing the graphics file with the NCAR meta-file viewer, idt, after the session.

fix_edge — identifies the beginning flux surface at which the p and F profiles are to be "rolled-off" smoothly to zero at the edge. The default value of ***fix_edge*** is 0.9 and will therefore modify p and F from $\bar{\psi} = 0.9 \rightarrow 1$. To use the EFIT profiles as-is, specify ***fix_edge*** = 0 (or 1); caution: finite values at the edge may impede Caltrans convergence.

make_inverse — when non-zero (the default), signifies that an inverse equilibrium is to be created (a direct-solve equilibrium is always created). If an inverse save-file is not desired, pass 0 to ***make_inverse***.

msrf — specifies the value of the Caltrans variable `msrf`—the number of flux surfaces to be used by the Grad-Shafranov solver.

thetac — specifies the value of the separatrix proximity factor θ_c defined below, and is used to convert the free-boundary solution to an acceptable fixed-boundary solution in the inverse solver; θ_c must be greater than zero if the free-boundary equilibrium is limited with an x-point and an inverse solution is to be constructed.

inv_k — profile option for the inverse solver (see §5.1)

inv_p — profile option for the inverse solver (see §5.1)

Normalized poloidal flux is:

$$\bar{\psi} \equiv \frac{\psi - \psi_{axis}}{\psi_{edge} - \psi_{axis}}, \quad 0 \leq \bar{\psi} \leq 1$$

where ψ_{axis} is the value of flux at the magnetic axis and ψ_{edge} is the value at which the plasma current and pressure go to zero. This limiting surface, may be due to a physical (or fictitious) limiter or an x-point. If the latter, the separatrix proximity factor, θ_c, determines where the edge of the plasma is relative to the separatrix:

$$\theta_c = \frac{\psi_{xpt} - \psi_{edge}}{\psi_{xpt} - \psi_{axis}}$$

Non-zero θ_c is necessary to construct a fixed-boundary solution from an x-point limited free-boundary equilibrium to avoid the discontinuity at the x-point.

Typical values of θ_c are $10^{-2} \cdots 10^{-3}$. Obviously, as θ_c is increased, the resulting equilibrium will increasingly deviate from the $\theta_c = 0$ free-boundary solution, but if θ_c is too small, errors can be introduced in subsequent calculations—particularly stability analyses. Finally, free-boundary solutions can be developed with finite θ_c to, for example, compare with fixed-boundary solutions.

Default values of `fix_edge`, `msrf` and `thetac` are subject to change; display the self-contained help message with `d3("help")` or just `d3` to get up-to-date information.

The d3 procedure now uses $\psi(R,Z)$ from the EQDSK file as an initial guess. This should aid convergence, but if a failure occurs, use $\psi(R,Z)$ from the generic save-file by setting use_eqdsk_psi=false before executing d3.

Finally, when the d3 procedure finishes, the free-boundary solution is restored (to be ready for another morph process). However, to operate with the inverse solution at that point, it must be restored (see §5.1).

3.2 Files created by d3

An equilibrium save-file and an inverse-equilibrium save-file (unless the flag make_inverse=0) will be created upon successful execution of the d3 procedure. These files will be named, respectively, *shot_time*.sav for the free-boundary equilibrium and *shot_time*_inv.sav for the inverse equilibrium, where the character strings *shot* and *time* are constructed from Caltrans character variable shotName and real variable shotTime. If an EQDSK file represents an actual shot and a time-slice, these quantities get assigned from the DIII-D shot number and its corresponding time-point. The user can also write save-files directly, with any name, as described in §7.

The graphics meta-file (ncgm file) will include EFIT-Caltrans p', FF' and q-profile comparisons, coil-current comparisons and configuration plots.

3.3 Morphing multiple equilibria

If several g-files are to be processed it is more efficient to use a Basis do-loop to execute d3. There are two helper routines, gfiles and afiles, defined in d3.bas, which aid in processing multiple EQDSK files. Typing gfiles with no argument will build a list of the g-file names in the current directory and return them in a character array named gfilelist. The gfiles routine is defined as a function—it returns the number of unique shot numbers represented in the current directory. If an individual shot number (or 0, implying all shot numbers) is given as an argument to gfiles, then a list of g-file time-points will be displayed.

The afiles procedure will display a one-line summary of the a-files in the current directory, showing the configuration code (limloc) and other information.

As an example, to morph all EQDSK files in the current directory into Caltrans equilibria, do something like the following:

```
caltrans d3.bas
gfiles
d3(gfilelist(1),0)
do $i=2,length(gfilelist)
   d3(gfilelist($i),0)
enddo
```

In this application (where `d3` is executed within a do-loop) it is necessary to call it once—prior to entering the loop—due to limitations in the Caltrans restore process.

Note the 2nd argument to `d3`, *`confirm`*, is zero in the above example so execution will proceed *without* displaying any graphics (be sure to peruse the `ncgm` file with `idt` after the session to verify the results, however). Also note the use of the chameleon variable `$i`, used here to avoid having to declare an integer loop variable.

4 Creating equilibria from scratch

The script `tokamak.bas` is available to help create equilibria for a new configuration. The script file defines a so-called dead-start procedure that creates a free-boundary equilibrium from a small set of parameters given in a text file.

The file listed in Appendix A is an example of input for the dead-start procedure. The first line contains a problem identification string ("Tokamak/DN" in this example) which is used to name the resulting equilibrium save-file. If is followed by four blocks of parameters for: (1) the plasma, (2) the toroidal field, (3) the computational grid and (4) plot scales. The plasma and toroidal field blocks are mandatory; default parameters for the last two blocks will be provided.

A set of PF coils will be automatically generated, unless a 2nd file is available, as described at the end of this section.

To create a free-boundary equilibrium (where PF coil specifications will be generated automatically), do the following:

```
caltrans tokamak.bas
ds("filename") # invoke the dead-start procedure
```

which, if successful, will create a file named (in this case) `tokamak_dn.sav`. The default input file name to `ds` is `tokamak.inp`.

This save-file may then be used to construct an inverse equilibrium as described in §5.

If a specific set of PF coils are known, create a file similar to the one listed in Appendix B using any file name (or use the default file name `pfcoil.inp`). Then execute the dead-start procedure with two input file names:

```
caltrans tokamak.bas
ds("tokamak.inp","pfcoil.inp")
```

where the file names shown here are the default names.

The equilibria created with the dead-start use model profiles and relatively crude shape specifications—they usually serve as a starting point for more detailed models.

5 Altering profiles and shape

Commonly needed tasks are changing the resolution of an equilibrium or varying profile quantities or the plasma shape, then quantifying the impact on MHD stability.

These can be performed with the direct or free-boundary solver; however they are more easily accomplished using the inverse, fixed-boundary solver, starting from some initial state in an inverse save-file. Inverse equilibria have the advantage of providing better resultion near the magnetic axis than direct-solve solutions.

Inverse save-files for DIII-D are most commonly created with the morphing procedure, `d3`, but can be created from any Caltrans free-boundary solution.

To create an inverse equilibrium from a free-boundary equilibrium, execute the `start_inv` routine to create an initial inverse equilibrium and save it to disk as follows:

```
caltrans name.sav
thetac=0.01; run
nht=200; epsrk=1.0e-06
start_inv
saveq("name_inv.sav")
```

where the inverse save-file can be used by the `teq_inv` routine, as described in the following sections. The `saveq` routine is described in §7.

When creating an inverse solution from a direct-solve equilibrium, the separatrix proximity parameter, θ_c, must be non-zero if the initial equilibrium is diverted. This is accomplished with the `thetac=0.01; run` statement. The parameters `nht` and `epsrk` are the iteration limit and convergence criterion for the inverse solver. The `start_inv` routine extracts profiles and shape from the free-boundary solution and then executes the inverse solver, `teq_inv`, described in the following sections.

The `start_inv` routine copies certain profile quantities to "save" variables for use as constraints in subsequent inverse solves, as described in the next few sections. These save-quantities may be modified by the user to alter an inverse equilibrium.

5.1 Inverse (fixed-boundary) solver

The inverse solver in Caltrans—a modified version of POLAR1—is invoked with the `teq_inv(inv_k,inv_p)` script function: an interface to the compiled subroutine `teqinv`. The arguments to the script function are optional—the default values are contained in global variables `inv_k` and `inv_p`. The first argument, `inv_k`, is an integer in {0,1,2,3} that specifies which profile quantities are to be preserved (i.e., used as input to the Grad-Shafranov equation). The 2nd argument, `inv_p`, specifies whether the plasma flux, toroidal current

or F_{edge} is to be constrained. Table 1 summarizes the constraint options available for the inverse solver, and the mapping to code variable names (for online documentation, do: `list teqinv`).

Table 1: `teq_inv` constrained profile options via `inv_k`.

`inv_k`	*constrains...*
`0`	$p \mapsto$ `psave` and $q \mapsto$ `qsave`
`1`	$p \mapsto$ `psave` and $FF' \mapsto$ `frsrf*fpsrf`
`2`	$p \mapsto$ `psave` and $\langle \mathbf{J} \cdot \mathbf{B} \rangle / \langle F/R^2 \rangle \mapsto$ `jtsave`
`3`	$p \mapsto$ `psave` and $\langle \mathbf{J} \cdot \mathbf{B} \rangle / \langle B^2 \rangle \mapsto$ `jparsave`

The entropy s may be constrained instead of the pressure: if `entropy_flag=1` then $s \mapsto$ `ssave`. The "`save`" quantities are reset, e.g., `qsave=qsrf`, etc. after a direct-solve solution.

The constraint options for the 2nd argument to `teq_inv`, `inv_p`, are summarized in Table 2.

Table 2: `teq_inv` constraints $\Delta\psi_p$, F_{edge} or I_p via `inv_p`.

`inv_p`	`inv_k=0,3`	`inv_k=1,2`
		$F^2_{edge} = F^2_{wall}$
`< 0`	$\Delta\psi_p$	$\Delta\psi_p$ and profiles scaled
`= 0`	F_{edge}	profiles not scaled
`> 0`	I_p	I_p and profiles scaled

The variable names for the plasma flux, vacuum F and toroidal current are: $\Delta\psi_p \mapsto$ `dpsi00` (reset from `dpsi0`) [G cm$^2/2\pi$], $F_{wall} = R_0 B_{\varphi,0} \mapsto$ `fwall` [G cm] (which gets its value from the `ro*btor`) and $I_p \mapsto$ `plcm` [MA]. Variables `ro` and `btor` are the reference radius [cm] and vacuum toroidal field [G], respectively.

Script function `teq_inv(inv_k,inv_p)` resets the *inactive* `save` variables, and its arguments default to the present value of variables `inv_k` and `inv_p`. The arguments to the complied routine `teqinv(inv_k,inv_p)` do not have default values, and the inactive `save` variables are not reset.

To begin operations with an inverse equilibrium, first load one into memory with the `restore` *`name`* statement[9] and execute the inverse solver.

```
restore "name_inv.sav"; teq_inv
```

[9] The Basis `restore` and `read` statements take a single filename as an argument. If you get an unexpected error message during one of these input operations, try quoting the filename.

Two parameters used by the inverse solver are the maximum number of iterations (code variable `nht`) and the convergence criterion (`epsrk`). The convergence criterion specifies the magnitude of the allowable residual—a reasonable value for the criterion is 10^{-6}. If the convergence criterion is not satisfied a warning message will be issued. The user has the option of increasing the allowable iterations or relaxing the convergence criterion. If the inverse solver is invoked in a script, something like the following is recommended.

```
call teq_inv
if (residj > epsrk) then
   <do something special>
endif
```

5.2 Changing the resolution

To alter the resolution of an inverse equilibrium, change the number of flux surfaces (`msrf`) and/or the number of poloidal points (`map`) and execute the `generate` routine which adjusts storage and interpolates to the new size(s):

```
msrf=128; map=128; generate
```

The total number of angle-like points is `mls=2*map-1` (`mls` is an output quantity). A plotting routine named `contour` can be used to show the flux surfaces and radials (see §6).

5.3 Plotting the q-profile

The q-profile is contained in vector `qsrf(1:msrf)` and dimensionless poloidal flux in `psibar(1:msrf)`. With a graphics window open, plot[10] the safety factor, $q(\bar{\psi})$:

```
plot qsrf,psibar
```

To plot q versus dimensionless toroidal flux, $q(\bar{\phi})$:

```
plot qsrf,phibar color=red
```

Therefore, to plot q versus the minor-radius-like quantity $a\sqrt{\bar{\phi}}$,

```
nf; plot qsrf,sqrt(phibar)*rbore color=rainbow
```

where `rbore` is the plasma minor radius, a, and `nf` is the Basis command to start a new graphics frame.

[10]The Basis EZN plot command has the syntax: `plot f(x),x [options]` which sometimes confuses those used to the opposite syntax: `plot x,f(x)` (as in IDL). Additionally, in Basis, the plot command can be issued with only the independent variable named, e.g., `plot f(x)`, in which case it will be graphed versus the vector of indices `1:length(f)` (if `f` is defined as a 1-origin array).

5.4 Altering the q-profile

As an example, construct an arbitrary shape factor that is unity near the magnetic axis and increases at the edge, say:

```
real foo=1 + 0.5*psibar**4
```

Now specify, in vector `qsave`, a *desired* q-profile:

```
qsave=qsrf*foo
```

and make a comparison plot of the initial and desired profiles:

```
nf; plot [qsrf,qsave],sqrt(phibar)*rbore
```

To modify the equilibrium, just execute the inverse solver:

```
teq_inv(0,-1)
```

where the first argument, `inv_k` $= 0$, specifies `qsave` as the input profile quantity for the Grad-Shafranov equation. Repeating the previous

```
nf; plot [qsrf,qsave],sqrt(phibar)*rbore
```

command, we see the equilibrium q-profile, `qsrf`, is now *approximately* identical to the desired `qsave`. To see the difference, enter `qsrf - qsave`, in which case all entries should be "small"—the differences should get smaller as the resolution is increased. The vector `qsrf` is calculated for all values of `inv_k` as a line-integral diagnostic on the final solution.

Executing `teq_inv(0,-1)` with the desired q-profile, as specified in `qsave`, also holds fixed: (1) the plasma boundary, (2) the pressure profile, and (3) the plasma flux:

$$\Delta\psi_p = \psi_{edge} - \psi_{axis}$$

The Caltrans variables that contain the preserved quantities of interest here are: `qsave`, the vector `psave` (initialized to the pressure `prsrf(1:msrf)` when the initial inverse equilibrium was created), and the constrained plasma flux `dpsi00` which is set by `teq_inv(0,-1)`to $\Delta\psi_p$ (code variable `dpsi0`), the enclosed flux of the most recently calculated equilibrium. The plasma current, in this example, therefore decreased.

5.5 Altering the ohmic current profile

The "ohmic" current profile, $J_o(\psi)$, (code variable `jtsrf(1:msrf)`) is:

$$J_o(\psi) \equiv \frac{\langle \mathbf{J}\cdot\mathbf{B}\rangle}{\langle \mathbf{B}\cdot\nabla\varphi\rangle}$$

and the vector `jtsave` is used by the inverse solver to preserve it. To change the ohmic current profile and preserve the plasma flux (which scales from the input `jtsave` and p'):

```
jtsave=jtsrf*foo
nf; plot [jtsrf,jtsave],sqrt(phibar)*rbore
teq_inv(2,-1)
```

To preserve the toroidal current, as given in code variable `plcm` [MA]:

```
teq_inv(2,1)
```

With this option the profile shape is conserved, but not its magnitude.

To let the current and plasma flux "float":

```
teq_inv(2,0)
```

5.6 Altering the parallel current profile

The parallel current profile, $J_\parallel(\psi)$, (code variable `jparsrf(1:msrf)`) is:

$$J_\parallel(\psi) \equiv \frac{\langle \mathbf{J} \cdot \mathbf{B} \rangle}{\langle B^2 \rangle}$$

and the code variable `jparsave` is the name of the preserved quantity. In a manner similar to modifying the ohmic current profile, conserving flux:

```
jparsave=jparsrf*foo
nf; plot [jparsrf,jparsave],sqrt(phibar)*rbore
teq_inv(3,-1)
```

and to modify $J_\parallel$ while holding the toroidal current to `plcm`:

```
teq_inv(3,1)
```

5.7 Altering the pressure profile

The pressure profile may be altered by changing the desired value of $p(\psi)$: `psave(1:msrf)` followed by executing `teq_inv` with any value of `inv_k`.

An alternate option is to preserve the entropy. Normally the entropy flag (code variable `entropy_flag`) is 0 and `psave` is used to set the pressure profile. If the entropy flag is 1 then the entropy vector `ssave` will be used.

5.8 Altering the plasma boundary

The plasma boundary used by the inverse solver typically comes from a prior free-boundary solution, where the POLAR1 boundary coordinates `uk` and `vk` are filled, respectively, with `rls` and `zls`—the coordinates from the free-boundary solver (all of these quantities are in units of centimeters).

The `rls(1:mls)`,`zls(1:mls)` coordinates are ordered CCW and close, while the `uk(1:ntet)`,`vk(1:ntet)` coordinates, also ordered CCW, *overlap*, so the proper mapping is:

```
uk(1) =rls(mls-1)
vk(1) =zls(mls-1)
uk(2:ntet)=rls(1:mls)
vk(2:ntet)=zls(1:mls)
```

where `ntet=mls+1`.

To alter the plasma boundary while in the inverse equilibrium mode, change the `uk`,`vk` entries and re-execute the inverse solver.

6 Caltrans graphics routines

There are a few Caltrans-specific plot routines which are defined as functions in a standard script-file that gets read automatically into each session. The most commonly used ones are: `layout`, `contour`, `profiles` and `pb`. The first two are appropriate for free- and fixed-boundary equilibria; the last two should be used only for free-boundary solutions.

The overall configuration can be displayed with the

`layout(`*`style`*`,`*`legend`*`)`

routine, where *`style`* is in {0,1,2} and *`legend`* in {0,1}. Style type 0 shows the PF coils with their true size and shape, style 1 draws the coil cross-sections in proportion to their current, and style 2 shows the coils with their true size and discretization. A table of parameters will be displayed if *`legend`* is non-zero (both arguments to `layout` default to 1).

The `contour` routine displays the current equilibrium solution in terms of flux-coordinates. This routine takes up to three arguments:

`contour(`*`last_surface`*`,`*`skip_srf`*`,`*`skip_radials`*`)`

where *`last_surface`* $\leq$ `msrf` is the index of the last flux surface to show, *`skip_srf`* is the number of surfaces to skip and *`skip_radials`* is the number of radials to skip. To see all surfaces and radials do `contour(msrf,1,1)`.

The `profiles` routine plots, for free-boundary equilibria, various profile quantities.

The `pb`, for "plot boundary", makes a terse plot of the limiting flux surface.

7 Save-files

7.1 Writing save-files

The Caltrans routine `saveq` writes selected equilibrium quantities to a binary file—called a save-file—using the PFB facility as described in §2.9.2. Save-file names must be have a `.sav` suffix, and, by convention, inverse-save-files have a `_inv.sav` suffix.

Saving a free-boundary equilibrium:

```
saveq("name.sav")
```

and saving a fixed-boundary (inverse) equilibrium:

```
saveq("name_inv.sav")
```

Save files created with either (or any) name contain identical variables; saved variable `inv_eq` will be 0 for free-boundary solutions and 1 for inverse solutions so when the files are restored they are treated appropriately.

7.2 Importing save-files

Save-files can be imported or restored into a Caltrans session by naming them on the command-line or within the session with the Basis `restore` command:

```
restore name.sav # or restore "name.sav"
```

A `run` command or `teq_inv` command must be issued after the restore to make all diagnostic quantities consistent with the saved quantities.

8 Writing EQDSK files from Caltrans

EQDSK files can be written from the Caltrans session by executing the `weqdsk` routine. If the code is in free-boundary mode, then a pair (g-file and a-file) of EQDSK files will be written with names of the form:

```
<prefix><shot_string>.<time_string><suffix>
```

where: *`prefix`* is a single character indicating the file type, *`shot_string`* is taken from code variable `shotName`, *`time_string`* is constructed from code variable `shotTime` and *`suffix`* is the string `_teq`.

The user can change the contents of `shotName` and `shotTime`, and also the suffix string; execute `weqdsk("help")` for details.

If the code is in fixed-boundary mode, a t-file can be written for Alan Turnbull's GATO code—the prefix character will be `t`.

Write an a-file and g-file for a direct-solve equilibrium (the default):

```
weqdsk # or weqdsk("ag")
```

or write just a g-file:

```
weqdsk("g")
```

Write a file GATO from an inverse-equilibrium:

```
weqdsk("t")
```

One may also write a-file and g-file pairs from an *inverse* equilibrium with the following procedure.

Write an a-file and g-file from an inverse equilibrium:

```
note=" " # or some meaningful comment
inv_k=0
teq_inv
get_vacflux
weqdsk
```

9 Stability analyses

9.1 Caltrans balloon routine

The ideal ballooning, Mercier and resistive interchange criteria can be quickly evaluated with the `balloon` routine. In either free-boundary or fixed-boundary mode, set the number of poloidal periods to use in the evaluation (`mperd` is the "number of times around"). Execute `balloon` and its corresponding plot routine to see the results:

Stability analysis with `balloon`:

```
mperd=25; balloon; plot_ball
```

The `plot_ball` routine plots the results, including the neoclassical bootstrap contribution[11] to the Rutherford island growth equation.

9.2 DCON

DCON low-n ideal stability analyses can be performed within the Caltrans session with either a free-boundary or fixed-boundary equilibrium in memory.

9.2.1 Preparing the equilibrium

Start-up with or restore the equilibrium of interest—for a free-boundary equilibrium:

```
restore name.sav; run
```

or for a fixed-boundary equilibrium:

```
restore name_inv.sav; teq_inv
```

If the code is in free-boundary mode and the equilibrium is limited with an x-point ($\theta_c = 0$ and no active limiter), set θ_c to a small number and re-execute the free-boundary equilibrium, e.g.,

```
thetac=0.001; run
```

to avoid a singular condition at the x-point. Alternatively, one could use the DCON parameter `psihigh` to truncate the analysis at $\bar{\psi}_{truncate} \mapsto$ `psihigh`.

[11] *à la* Hegna (without polarization), circa 1997.

9.2.2 Executing DCON

DCON uses the current equilibrium in memory as input, as well as its own control parameters, so executing with default settings is quite simple.

Executing DCON for, say, the $n = 2$ mode with other parameters defaulted:

```
nn=2; dcon
```

Most of the relevant DCON parameters are defined in the group `Dcon_control`; `list Dcon_control` will display them. Refer to the users' guide[12] for details. Some of the more important input parameters are listed below. In some cases (e.g., `dcn.a`) it will be necessary to reference a DCON variable with its qualified name to resolve name conflicts.

`nn` — toroidal mode number, n

`a` — conformal wall parameter. For $0 \leq a < 10$ a conformal conducting wall is located at distance a from the plasma edge, normalized to the plasma radius. If $a \geq 10$, then the wall is treated as if at infinity (`dcn.a` is initialized to 20).

`mspi` — number of flux surfaces, fit[13] with cubic splines

`mtheta` — number of angles for generating poloidal harmonics

`mthvac` — number of angles for Morrell Chance's `VACUUM` package

`delta_mlow` and **`delta_mhigh`** — the increments applied to the poloidal mode number range (m_{low} and m_{high}). For fixed-boundary modes, $\Delta m_{low} = \Delta m_{high} = 0$ works well and increasing them increases the execution time without improving the accuracy. For free-boundary modes they should be positive and large enough to accommodate the bandwidth of the equilibrium shape. In general, they should be varied until reasonable convergence is attained.

`sing_start` — the number of the singular surface at which the radial integrator starts with a zero-flux boundary condition. This option is used for moderate to high n if the mode is localized near the edge, for peeling modes or ballooning modes. If accurate relative MHD energies are needed, it will be necessary to test the sensitivity to various input parameters. DCON is an energy-principle code in the number of poloidal harmonics. On the other hand, if one is only interested in the existence of an instability, the number of harmonics need not be increased if one is found. However, this conclusion must be "converged" relative to the spatial resolution.

[12] As of this writing, Caltrans is loaded with DCON version 3.60. Glasser's `README` file is available under `/d/osf/dcon/dcon_3.60/` at GA and under `/mfe/theory/Dcon/dcon_3.60/` at LLNL.

[13] DCON uses `lsode` to integrate the radial equations so the stability code is "gridless" in radius.

9.2.3 DCON accuracy diagnostic

In addition to the summary printed output, the output arrays `di(1:lsing)` and `di0(1:lsing)`, where `lsing` is the number of singular surfaces found by DCON, should be inspected after each execution. The vector `di` is a matrix eigenvalue which should equal the Mercier criteria, contained in `di0`, to within a small difference. In Basis parlance, do:

```
max(abs((di/di0 - 1))
```

If the maximum relative difference is greater than, say, 10^{-2}, consider increasing the resolution. Graphical display of `gsei.bin` data with `xdraw`, as described below, is also useful.

9.2.4 DCON graphical output

Caltrans makes a symbolic link in the current working directory to the DCON graphics viewer `xdraw` so viewing the DCON graphical output (which is separate from the Caltrans `ncgm` file) can be performed in a sub-process controlled from the Caltrans session, e.g.:

```
!xdraw dcon # profiles and local stability
!xdraw crit # internal stability results
!xdraw gsei # accuracy information
```

A summary of `xdraw` commands will be displayed in response to keying a `k` character while the focus is over an `xdraw` graphics window—key in `q` to close the `xdraw` windows.

9.2.5 Executing DCON with a wall

The parameter `dcn.a` can be used to vary the position of a *conformal* (to the plasma boundary) perfectly-conducting wall over the range $0 < a < 10$, where a is the plasma-wall distance, normalized to the plasma radius. In addition, an arbitrarily shaped wall can be created with the `sewall` routine, as described below.

The auxiliary routine `sewall` (for smoothed experimental wall) can be used to impose an arbitrary wall shape for DCON's VACUUM module. The `sewall` routine takes the geometric definition of the wall from one of three models: (1) the passive structure model, described with EFIT-style parallelogram elements and contained in the current save-file (which for DIII-D describes the vacuum vessel, ADP ring and upper divertor supports), (2) the first-wall model contained in the current save-file (the `rplate`, `zplate` arrays), or (3) a user defined set of coordinates contained in arrays `rwall`, `zwall`. The `sewall` routine is defined in script file `sewall.bas`, and its usage is given below. There are optional arguments to `sewall`; see the help message with `sewall("help")` for details.

Using an experimental wall geometry in DCON:

```
read sewall.bas
call sewall
dcon
```

The default wall model is now (with Revision 2) the vacuum-vessel elements from the passive structure model (previously it was the first-wall model). Global variable `usePassiveStructure`, initialized to `true`, may be set to `false` to select the plate (first-wall) elements. If `rwall` and `zwall` exist, the 3rd option will be used irrespective of the contents of `usePassiveStructure`.

The model used by GATO (defined in file `inwd3dnew_vessel`) can also be used by loading a file containing `rwall`, `zwall` coordinates, as follows.

```
restore d3d_vv_gato.pfb
call sewall
```

The GATO first-wall tile geometry is also available in file `d3d_fw_gato.pfb`, which contains the coordinates from GATO file `inwd3dnew_tiles`.

After installing a GATO model, use the Basis `forget` command to return to the default model:

```
restore d3d_vv_gato.pfb
call sewall
dcon # Do analysis with GATO model
forget rwall, zwall
call sewall
dcon # Do analysis with VV from passive structure model
```

Since the wall geometry may contain convolutions and sharp bends, they are, by default, smoothed by `sewall`. Options are available to change the degree of smoothing, or eliminate it entirely. The smoothed contour may also be scaled to change the relative wall position. Again, execute `sewall("help")` for details regarding the smoothing and scaling options. In addition to providing the smoothed wall geometry, `sewall` sets the DCON flags to trigger its inclusion, as the default wall model is conformal.

10 Demonstration script

A script file named `d3d_demo.bas` (listed in Appendix C) is distributed with Caltrans. It demonstrates the code features described in this document with a particular EQDSK file set (shot 106535, $t = 2.11$ s). To use it, start-up the code with only the demo filename on the command-line (it takes a few minutes to complete).

Running the demonstration script:

```
caltrans d3d_demo.bas
```

The new user is encouraged to copy the demonstration script from the Caltrans distribution and use it as a starting point for a personalized script.

When the default demo script has terminated, the following files will have been created by `caltrans`:

```
106535_2110.sav
106535_2110_inv.sav
d3d_demo.001.cgmlog
d3d_demo.001.ncgm
d3d_demo.log
d3d_demo_128_inv.sav
d3d_demo_inv.sav
```

Sadly, the `dcon` execution will have created an additional 20+ files, but they can be useful for viewing graphical output after the Caltrans session has ended (or running a stand-alone version of DCON).

Appendix

A Sample input file for the dead-start procedure

The text file listed below is an example of input for the dead-start procedure `ds` defined in script file `tokamak.bas`.

The first line contains a quoted problem identification string which is used to name the resulting equilibrium save-file. All of the items in the "Plasma" category must be specified, as well as the "Toroidal field". The `Dsep` entry is the radial separation, measured at the outboard midplane, between the active and inactive separatrices for a single-null equilibrium, so for a DN configuration it is zero. If a specific external flux linkage is desired, it can be specified or if zero is entered then the flux linkage will not be constrained.

The "Computational grid" and "Plot Scales" blocks may be omitted—defaults will be supplied.

```
"Tokamak/DN"

Plasma...
      1.00 MA       plasma current
      1.00 m        major radius
      0.50 m        minor radius
      0.00 m        Zaxis
      1.90          95% elongation
      0.30          95% triangularity
      0.00 m        Dsep
      0.50          poloidal beta
      0.80          li
      0.00 Wb       External flux linkage

Toroidal field...
      1.00 T @ R = 1.00 m

Computational grid (optional)...
      0.40 m        Rmin
      1.60 m        Rmax
     -1.20 m        Zmin
      1.20 m        Zmax
   33 x 65          No. grid points (Nr x Nz)

Plot Scales (optional)...
      0.00 m        Rmin
      2.50 m        Rmax
     -2.00 m        Zmin
      2.00 m        Zmax
      2.00 MA/m^2   Current density for drawing coil cross-sections
```

B Sample coil specifications for the dead-start procedure

The text file listed below is an example of PF coil input for the dead-start procedure defined in script file `tokamak.bas`.

The number of turns (`n_turn`) and the current and field capability limits (`NI_cap` and `B_cap`) are not used by Caltrans, but are sometimes used by user scripts and therefore fictitious values need be supplied.

```
"Tokamak"
 14 coils
name    Rc [m]   Zc [m]  DRc [m]  DZc [m]   n_turn    NI_cap  B_cap
"PF1U"  0.5610   0.2470   0.2135   0.4764        1       1       1
"PF2U"  0.5610   0.6932   0.2135   0.3808        1       1       1
"PF3U"  0.5610   0.9960   0.2135   0.1896        1       1       1
"PF4U"  0.5610   1.2510   0.2135   0.2852        1       1       1
"PF5U"  1.0850   2.2960   0.3330   0.3808        1       1       1
"PF6U"  3.0900   1.9200   0.1896   0.3808        1       1       1
"PF7U"  3.7300   0.9600   0.1418   0.2852        1       1       1
"PF1L"  0.5610  -0.2470   0.2135   0.4764        1       1       1
"PF2L"  0.5610  -0.6932   0.2135   0.3808        1       1       1
"PF3L"  0.5610  -0.9960   0.2135   0.1896        1       1       1
"PF4L"  0.5610  -1.2510   0.2135   0.2852        1       1       1
"PF5L"  1.0850  -2.2960   0.3330   0.3808        1       1       1
"PF6L"  3.0900  -1.9200   0.1896   0.3808        1       1       1
"PF7L"  3.7300  -0.9600   0.1418   0.2852        1       1       1
```

C The demonstration script d3d_demo.bas

```
chameleon d3d_demo_id = "$Id: d3d_demo.bas,v 1.2 2003/01/24 21:30:02 bulmer Exp $"

################################################################################
#                                                                              #
# Demo script to:                                                              #
#                                                                              #
#  (1) morph an EFIT DIII-D equilibrium,                                       #
#  (2) change profiles and obtain new inverse equilibrium, and                 #
#  (3) execute DCON.                                                           #
#                                                                              #
# See http://web.gat.com/caltrans/ for links to documentation, particularly:   #
#                                                                              #
#   o Setting up your Unix environment to use Caltrans                         #
#   o Documentation for Basis                                                  #
#   o Documentation for this script                                            #
#                                                                              #
################################################################################
#                                                                              #
# To execute this script, enter the following at the Unix shell prompt:        #
#                                                                              #
#         caltrans d3d_demo.bas                                                #
#                                                                              #
# The script will perform calculations, send graphics to an X-window and pause #
# until the user enters RETURN (or, when DCON frames are displayed with xdraw, #
# enter "q" with the keyboard focus in the grahics window).                    #
#                                                                              #
################################################################################


# (1) morphing an EFIT DIII-D equilibrium ######################################

# Read the morphing and "smoothed experimental wall" scripts into the session:

        read "d3.bas"
        read "sewall.bas"

# To see the help message for the morphing procedure:

        d3 # or d3("help")

# Morph an equilibrium. The 1st argument to d3 is the g-file name. The g-file
# should be in your current working directory (or in your Basis path). It is
# advisable to have the a-file available also, otherwise the procedure might
# not match the EFIT equilibrium very well, or have trouble converging.

        d3("g106535.02110")

# The d3 procedure, by default, will display graphical comparisons of EFIT and
# Caltrans equilibrium quantities. Enter RETURN after viewing each frame. If
# there are significant differences, report them to:
#
#       caltransbugs@alvin.llnl.gov
#
# with enough information to reproduce the problem.

# The save-files created by d3 are:

        call basisexe("ls -l 106535_2110*.sav")

# and can be used in the future to avoid having to re-morph them.

        echo=yes # to echo lines in this file as they are processed

# (2) Changing the plasma profiles #############################################

# (2.1) Changing the q-profile profile #########################################
```

```
# Here we want to work with inverse equilibria, so restore one into this
# session:

        restore "106535_2110_inv.sav"

# and run the inverse solver, to make everything consistent:

        teq_inv(0,-1)

# To plot something, open a graphics window and issue a plot command of the
# form "plot f(x),x" (see the Basis EZN document).  Here we'll plot q versus
# normalized poloidal flux.  These quantites are in variables qsrf(1:msrf) and
# psibar(1:msrf), where msrf is the number of flux surfaces. You can use the
# "list" command to display builtin documentation:

        list qsrf

# So, to plot q versus psibar:

        ow("d3d_demo")
        nf # new frame command
        plot qsrf,psibar
        pause

# Create an arbitrary profile modification factor:

        real foo=1 + 0.5*psibar**4

# New vector "foo" varies from 1 at the magnetic axis to 0.8 at the edge.
# We use it to specify a q-profile "qsave" to be preserved by the inverse
# solver:

        qsave=qsrf*foo

# We'll compare the present (qsrf) and desired (qsave) profiles, plotting them
# versus radius using the normalized toroidal flux "phibar" and plasma minor
# radius "rbore":

        nf; plot [qsrf,qsave] sqrt(phibar)*rbore color=rainbow
        pause

# Compute a new inverse equilibrium with the desired q-profile, which also
# conserves the plasma flux "dpsi0":

        teq_inv(0,-1)

# Plotting the 2 quantites again shows that "qsrf" is *approximately* the same
# as "qsave":

        nf; plot [qsrf,qsave] sqrt(phibar)*rbore color=rainbow
        pause


# (2.2) Changing the "ohmic" current profile, jtsrf ############################


# Apply a modifying factor to the ohmic current:

        jtsave=jtsrf*foo
        nf; plot [jtsrf,jtsave] sqrt(phibar)*rbore color=rainbow
        pause

# (2.2.1) calculate inverse equilibrium by conserving the plasma flux (this is
# done by scaling from the input jtsave and pressure):

        teq_inv(2,-1)
```

```
# (2.2.2) by specifying the desired toroidal current in "plcm" [MA]:

        plcm=2
        teq_inv(2,1)

# (2.2.3) let the flux and current float:

        teq_inv(2,0)

# (2.3) changing the parallel current, jparsrf ###################################

        jparsave=jparsrf*foo
        nf; plot [jparsrf,jparsave] sqrt(phibar)*rbore color=rainbow
        pause

# (2.3.1) conserving flux:

        teq_inv(3,-1)

# (2.3.2) conserving current, plcm (again by scaling input profiles):

        plcm=2.2
        teq_inv(3,1)

# Any of the above inverse equilibria can be saved in a portable binary file
# by invoking the "saveq" routine:

        saveq("d3d_demo_inv.sav")

# To change the resolution of the inverse equilibrium, change the number of flux
# surfaces "msrf" and/or the number of poloidal points "map" and execute
# "generate".  The total number of poloidal points is mls=2*map-1.  The
# contour routine can be used to see the flux surfaces:

        msrf=128; map=128; generate
        contour
        pause

# Save this one in a new portable binary file.

        saveq("d3d_demo_128_inv.sav")

# (3) MHD stability ##############################################################

# (3.1) with the balloon routine #################################################

# Specify the number of periods to evaluate with, execute the stability
# routine and plot the results (Mercier, resistive interchange, neoclassical
# bootstrap and ballooning criteria):

        mperd=25; balloon
        plot_ball
        pause


# (3.2) low-n criteria with DCON #################################################

# (3.2.1) Specify the toroidal mode number and execute DCON:

       nn=1; dcon

# (3.2.2) Now execute with a wall near the plasma:

       dcn.a=0.1; dcon

# (3.2.3) Use the DIII-D vessel. Having read the "smoothed experimental wall"
# script, execute the sewall routine, then DCON:
```

```
       call sewall
       pause
       dcon

# (3.2.4) Execute DCON's graphics command, xdraw (can be used after any DCON
# execution). To quit xdraw, enter "q" with the keyboard focus over one of the
# graphics windows, or enter "k" for the xdraw help package.

# DCON accuracy information:

       call basisexe("xdraw gsei")

# Equilibrium profiles and local stability:

       call basisexe("xdraw dcon")

# Low-n internal stability:

       call basisexe("xdraw crit")

################################################################################

       quit # end session
```