TEQ Demonstration

Topics:

- Code start-up (from save-file or dead-start)
- Making changes and saving them
- Graphics and output
- Modifying PF coils
- Managing configuration changes (à la ITER)
- Equilibrium constraints (shape, flux, coil currents)
- Analytic profile models, modifying profiles
- Writing EQDSK files
- Importing EFIT solutions via EQDSK
- Constrained equilibrium solver, `ceq`
- l_i-survey example
- Fiducial-state creation
- DIII-D Demo (morphing, inverse eq, balloon, DCON)

TEQ Packages

The TEQ free-boundary equilibrium solver may be accessed by one of three packages:

`eq` : Equilibrium solver (default)
`ceq` : Constrained equilibrium solver
`meq` : Equilibrium solver with minimization

Example:

```
caltrans save-file-name
package ceq
```

Code Start-up

Usually with a previously created “save-file”:

```
caltrans iter.sav
```

which may be a private or public file.

Sometimes a “dead-start” is necessary:

```
caltrans tokamak.bas
ds
```

http://wormhole.ucllnl.org/software/corsica/tokamak_ds.pdf

Saving Stuff

Change something and save it:

```
caltrans iter.sav
list eq.rc  # documentation for variable rc
rc(1) = rc(1) + 0.5
run
saveq(“my-iter.sav”)
quit
```

later on:

```
caltrans my-iter.sav
rc  # display contents of rc
```

Graphics and Output

Corsica automatically loads two script files (`ploteq.ezn` and `graphics.bas`) which contain many graphics “commands”, like `layout`, `profiles`, etc. These commands are written with Basis graphics package EZN (interface to NCAR graphics library).

Users can (a) execute the predefined Corsica graphics routines, (b) plot data directly using EZN features or (c) write their own routines.

Get the Basis documents at https://wci.llnl.gov/codes/basis/

Graphics Examples

Note for IDL users: the Basis plot command assumes the 1st argument is the *dependent* variable and the 2nd argument is the *independent* variable...

```
plot y x  # opposite to IDL's order

plot y  # also works (y versus its index)

plot y, x color=red  # commas optional...

plot y x mark circle  # as are equal signs
```

Corsica (Basis) I/O

Read scripts (written in the Basis language) with the read command:

```
read script-file.bas
```

Basis stream I/O facility:

```
# Read data...
integer io = basopen(file-name, “r”)
real x
io >> x
# Write data...
io = basopen(“somefile2.txt”, “w”)
```

Corsica I/O (cont.)

```
io << x
basclose(io)
```

To read, say, a file containing a table of 1000 paired x,y values...

```
real buf(2, 1000)
integer io = basopen("table.txt", "r")
io >> buf
basclose(io)
real x = buf(1,)
real y = buf(2,)
```

Corsica I/O (cont.)

Formatted output...

```
integer i, io = basopen("table.txt", "w")
do i = 1, length(x)
   io << format(x(i), 8, 2, 1) \
      << format(y(i), 12, 6, 1)
enddo
basclose(io)
```

There are routines in standard script file `text_util.bas` to facilitate reading formatted data, but Basis stream input usually does the job.

Binary I/O

Efficiently read and write data with the Basis portable database facility, PDB, aka PFB. Read the “PFB Package” chapter in the Basis Library manual.

Create a PDB:

```
create file.pfb
write var1, var2, ...
close
```

Restore a PDB:

```
restore file.pfb
list var1
```

Binary I/O (cont.)

Open a PDB for inspection:

```
open file.pfb
ls  # List contents
close
```

Corsica save-files are just PDB files with a *particular* set of variables. The `.sav` suffix, when appearing on the command-line, tells Corsica to restore the contents of the PDB into memory *and* try to execute an equilibrium calculation with it.

Within a Corsica session you must execute a `restore` command on a `.sav` file then a `run` command to make the equilibrium.

Modifying PF Coils

Deleting coils

Number of coils: `nc`, number of driven coils: `npfc` (< `nc`) and number of coils shown in plots: `ncplot` (< `nc`).

To omit the last coil:

```
nc = nc – 1; npfc = nc; ncplot = nc; run
```

To delete the ith coil:

```
pfid = [pfid(1:i-1), pfid(i+1:nc)]
```

Modifying PF Coils (cont.)

```
rc = [rc(1:i-1), rc(i+1:nc)]
```

etc. for *all* coil specifications (`pfid`, `rc`, `zc`, `drc`, `dzc`, `ac`, `ac2`, `nrc`, `nzc`, `cc` and `ic`), but remember that `ic` entries must not contain any “gaps”, so generally:

```
ic = [ic(1:i-1), ic(i+1:nc) - 1]
ic = where(ic < 0, 0, ic)
```

and finally:

```
nc = nc – 1; npfc = nc; ncplot =nc; run
```

Modifying PF Coils (cont.)

To add a coil, first increment the counters then insert new specifications where desired:

```
nc = nc + 1; npfc = nc; ncplot = nc
rc = [rc(1:i), rc_new, rc(i:nc-1)]
```

etc., but be careful with `cc` and `ic`:

```
cc = [cc(1:i), 1e-6, cc(i+nc-1)]
ic = iota(nc)
run
```

Managed Configuration Changes

Create one or more input files*, read script `device.bas`, then execute read_device:

```
caltrans iter.sav device.bas
read_device
run
saveq(“new-iter.sav”)
```

* File set (documentation in work):

`coils.in`, `fwall.in`, `limits.in`, `params.in`, `passive.in`, `shape.in`, `tfcoil.in`

Equilibrium Constraints

Plasma shape constraints:

- Fixed R-Z points: `rbd`, `zbd (1:nbd)`
- “Fuzzy points” w/wts: `rfbd`, `zfbd`, `alfbd (1:nfbd)`
- Limiter point: `rlim(0)`, `zlim(0)`

Flux linkage constraints:

- None (`vltf` = `cejima` = 0)
- Absolute: specify $<\Psi_{ext}>$ (`vltf`, `cejima = 0`)
- Relative: specify Ψ_0 and C_{Ejima} (`vltf` & `cejima`)

Equil. Constraints (cont.)

Separatrix separation for SN configurations:

- Relative flux difference between upper and lower x-point surfaces (`irl` = 0, `rl` = $(\Psi_{upper} - \Psi_{lower})/\Delta\Psi_p$)
- Difference in outboard major radius at $Z = Z_{axis}$ crossings of the upper and lower x-point surfaces (`irl` = +1 or -1, `rl` = ΔR_{sep})

Do "`list rl`" for details.

Note: variable `dsep` contains the actual separatrix separation.

Coil Current Regularization

Selector `ircwt` determines the regularization scheme (-1, 1, 2, …, 11) in evaluating coil currents `cc` where `ic(i)` ≠ 0 for the ith coil; `ic(i)` = 0 fixes the ith coil current at its present value.

`ircwt` = 1 minimizes: || `cc` – `cc0` ||
`ircwt` = 2 minimizes: || `cc` ||
`ircwt` = 3 minimizes: || J_{coil} ||

Vary `alfbd` weights to determine “best” plasma-shape/coil-current trade-off for your application.

Plasma Profiles

The two free flux-functions in the ideal MHD Grad-Shafranov toroidal equilibrium equation are the pressure and poloidal current functions, $p(\psi)$ and $F(\psi) = RB_\varphi$.

A collection of model profile forms are available in Corsica to specify p and F. They are expressed in terms of normalized poloidal flux in the plasma, x:

$$x = \frac{\psi - \psi_{axis}}{\psi_{edge} - \psi_{axis}}$$

which is held in variable `psibar`.

Plasma Profiles (cont.)

Plasma profile models are specified by setting a p-selector (`ipp`) and an F-selector (`ipf`). These select the analytic forms to be used.

Another selector determines how F is to be calculated: *directly* from the profile form or *indirectly* from an "ohmic current" profile form.

```
list ipp; list ipf; list ipj
```

Plasma Profiles (cont.)

Most of our tokamak *design-equilibrium* modeling uses:

```
ipj = 2; ipf = ipp = 3
```

i.e., we are specify an ohmic current profile (calculating F indirectly) and the pressure and J_0 functions have the form:

$$J_0 \propto (1 - x^{b_0})^{a_0} \quad \text{and} \quad p \propto (1 - x^{b_1})^{a_1}$$

where, for J_0, a_0 is `alfa(0)`, b_0 is `betp(0)` and for p, a_1 is `alfa(1)` & b_1 is `betp(1)`.

Plasma Profiles (cont.)

Note:

1. Parameter `betaj` (> 0) is a scale-factor which controls the magnitude of the pressure.
2. The G-S solution (with `ipscl` = 0) is scaled for toroidal current I_p, input parameter `plcm` [MA].

Plasma Profiles (cont.)

Profiles with edge pedestals are created by superimposing an edge contribution on to the parabolic form (`ipf`, `ipp` = 3).

The pedestal parameters are magnitude ε and exponent n yielding the J_0 profile form:

$$J_0 \propto (1-x^b)^a + \epsilon C (1-x) x^n \text{ with } C = \frac{1}{n^n}(1+n)^{(1+n)}$$

In Corsica, $\varepsilon \rightarrow$ `epf` and $n \rightarrow$ `-npf` (note the sign of `npf`!) for J_0 and `epp` and `-npp` for pressure.

Writing EQDSK Files

Script function `weqdsk` writes a, g, and some other EQDSK-like files. Execute with:

```
weqdsk(type, suffix, time_units, fw)
```

where *`type`* is “`a`”, “`d`”, “`g`”, “`i`”, or “`t`”. Default values are provided for all arguments. See the help message:

```
weqdsk(“help”)
```

Reading EQDSK files

Read a or g-EQDSK files with `reqdsk` (an interface to `reqdska` and `reqdskg`).

These routines read the contents into global variables with names of the form `a_<`*`name`*`>` or `g_<`*`name`*`>`, where `<`*`name`*`>` is the EFIT variable name. Again, do:

```
reqdsk("help")
```

Importing EFIT Equilibria

Since the contents of EQDSK files are not universally defined, special-purpose scripts are required to import an EFIT equilibrium (e.g., `cmod.bas`, `d3.bas`, `nstx.bas`) into Corsica.

These routines read a- and g-EQDSK files, and are executed with something like:

```
caltrans d3.bas  # No save-file required
d3(gfilename)
```

There is also an `mdsd3.bas` which reads data from an MDSPlus EFIT tree.

Constrained Equil. Solver

Package `ceq` is an interface to Michael Powell's hybrid method (called HYBRD) for finding the roots of a system of nonlinear equations. Do "`package ceq`", then pose a problem: say we want to vary the `betaj` parameter to achieve a specific value of β_p, say 0.5:

```
nctot = 1          # Number of constraints
vo = "betap(1)"    # Identify the constraint
vo0 = 0.5          # Desired value
vi = "betaj"       # Independent variable
x0 = betaj         # Initial guess (say, present value)
ihy = 20           # Limit on HYBRD iterations
run                # Execute HYBRD which will call G-S
                   #   solver as needed
```

CEQ (cont.)

A common `ceq` problem is solving for `betaj`, `alfa(0)` and `betp(0)` for desired values of β_p, l_i and *q(0)*, e.g.:

```
nctot = 3
vo = [“betap(1)”, “li(3)”, “qsrf(1)”]
vo0 = [0.5, 1.0, 0.9]
vi = [“betaj”, “alfa(0)”, “betp(0)”]
x0 = [betaj, alfa(0), betp(0)]
ihy = 20
run
```

CEQ (cont.)

If constraints are expressions:

```
iequa = -2  # To parse and evaluate vo
vo = “max(cc)”
```

If any coil diagnostics (`pffz`, `pfbc`, `ufc`, etc.) are used as constraints, set:

```
lop0 = 1  # Turn on coil diagnostics, and
kbfc = 1  #   evaluate at each iteration
```

CEQ (cont.)

Create an ITER EOB state:

```
package ceq
lop0 = 1 ; iequa = -2 ; kbfc = 1
vltf = vltsnd(1); cejima = 0
ic(7:12) = [7, 0, 8, 8, 0, 9]
nctot = 6
vo  = ["betap(1)", "li(3)", "qsrf(1)", \
       "max(ufc(9:10))", "csfz_rep", "cc(11)/cc(8)"]
vo0 = [0.65, 0.9, 0.9, 1, 120, 1]
vi  = ["betaj", "alfa(0)", "betp(0)", \
       "vltf", "cc(8)", "cc(11)"]
x0  = [betaj, alfa(0), betp(0), vltf, cc(8), cc(11)]
ihy = 40 ; run
run
```

CEQ (cont.)

An l_i-survey at ITER start-of-flattop involves a scan of l_i values, say 0.7 to 1.2, where $q(0)$ is unconstrained *unless* it drops below 0.9 where it is constrained to 0.9, but at high l_i this is unfeasible so $q(0)$ must fall below 0.9.

```
# File: li-survey.bas
# Start-up with the reference SOF equilibrium
chameleon basename = trim(probid)

# ceq settings...
package ceq
```

l_i-survey (cont.)

```
nctot = 2
vo = ["li(3)", "betap(1)"]
vo0 = [ li(3), 0.1]
vi = ["alfa(0)", "betaj"]
x0 = [alfa(0), betaj]
betp(0) = 1
ihy = 99; factor = 0.01; lop0 = 1

# Storage for results...
real zli = 0.01*fromone(iota(70, 120, 5))
integer i, n = length(zli)
real zq0(n), zufc6(n), zufc9(n)

# Survey loop...
logical addq0 = false, dropq0 = false
win
```

l_i-survey (cont.)

```
do i = 1, n
  vo0(1) = zli(i)
  probid = basename//" li="//format(vo0(1), 0, 2, 1)
  probid
  run
  if (qsrf(1) < 0.9 & ~addq0) then
    <<return<<"Adding q(0) constraint for" \
      <<trim(probid)
    addq0 = true
    nctot = 3
    vo(nctot) = "qsrf(1)"
    vo0(nctot) = 0.9
    vi(nctot) = "betp(0)"
    x0(nctot) = betp(0)
    run
  elseif (betp(0) > 4 & ~dropq0) then
```

l_i-survey (cont.)

```
      <<return<<"Dropping q(0) constraint for" \
        <<trim(probid)
      dropq0 = true
      nctot = 2
      run
    endif
    layout; profiles; pufc
    chameleon sname = "sof-li="
    sname = sname//format(vo0(1), 0, 2, 1)//".sav"
    saveq(sname)
    zq0(i) = qsrf(1)
    zufc6(i) = ufc(6)
    zufc9(i) = ufc(9)
  enddo
```

CEQ (cont.)

Script file `fiducials.bas` (under development) contains several routines to construct fiducial-state equilibria (IM, SOP, SOF, SOB, SOF, EOB). These routines, named `make_im`, `make_sof`, `make_fiducials`, etc. demonstrate the use of CEQ to solve a variety of constrained equilibrium problems.

CEQ (cont.)

Numerical accuracy and tolerances:

G-S solver declared “converged) when `residj` < `epsj` where `residj` is the relative change in mesh current from previous iteration and `epsj` is an input parameter.

CEQ solver (i.e., HYBRD) assumes object functions accurate to `epsfcn` and declares convergence when the relative error between iterates is less than `tol`. Parameter `factor` determines determines the bound on step size.

DIII-D Demo

Standard script `d3d_demo.bas` demonstrates several Corsica capabilities:

- “Morphing” an EFIT equilibrium in EQDSK files
- Creating an inverse equilibrium
- Modifying an inverse equilibrium
- Executing the balloon stability analysis routine
- Executing DCON as a Corsica package with various wall geometries