A User Guide
to the
PROCESS Fusion Reactor Systems Code

P. J. Knight

M. D. Kovari

Culham Centre for Fusion Energy
Culham Science Centre, Abingdon, Oxon, OX14 3DB, UK
## Contents

1 **Introduction** 9

1.1 Rationale ................................................................. 9
1.2 History ........................................................................... 9
1.3 Sources of information .................................................... 10

2 **Program Overview — The Fundamental Concepts** 11

2.1 Equation Solvers ........................................................... 11
  2.1.1 Non-optimisation mode ............................................. 11
  2.1.2 Optimisation mode ................................................... 11
  2.1.3 Scans ....................................................................... 15
2.2 The Variable Descriptor File .............................................. 15
2.3 Input Parameters ............................................................ 15
2.4 Iteration Variables ........................................................ 15
2.5 Constraint Equations ...................................................... 19
  2.5.1 Consistency equations ............................................... 19
  2.5.2 Limit equations ....................................................... 19
2.6 Figures of Merit .............................................................. 22
2.7 Scanning Variables ........................................................ 22

3 **Physics, Engineering and Other Models** 25

3.1 Tokamak Power Plant ....................................................... 25
  3.1.1 Radial and vertical build ............................................ 25
  3.1.2 Plasma physics models ............................................. 28
  3.1.3 Armour, first wall and breeding blanket ......................... 39
  3.1.4 Shield ....................................................................... 42
  3.1.5 Divertor ................................................................... 42
  3.1.6 Toroidal field coils ................................................... 42
  3.1.7 Poloidal field coils .................................................... 45
4 Running PROCESS

4.1 Environment set-up ................................................................. 69
4.2 User Interface .............................................................................. 69
4.3 Executing the Code ...................................................................... 71
4.4 The Input File ............................................................................. 71
4.5 The Output Files .......................................................................... 71
4.6 Optimisation mode ...................................................................... 71
4.7 Non-optimisation mode ............................................................... 72
4.8 Troubleshooting ........................................................................ 72
  4.8.1 Error handling ...................................................................... 73
  4.8.2 General problems .................................................................. 73
  4.8.3 Optimisation problems .......................................................... 73
  4.8.4 Unfeasible results ................................................................. 74
  4.8.5 Hints ................................................................................. 75

5 Changing the Source Code: New Models, Variables and Constraints 76

5.1 Source Code Modification .......................................................... 76
  5.1.1 Changing the Fortran code ...................................................... 76
  5.1.2 Source code documentation .................................................. 77
5.2 Input Parameters .......................................................................... 78
5.3 Iteration Variables ....................................................................... 78
5.4 Other Global Variables ............................................................... 79
5.5 Constraint Equations ................................................................... 79
5.6 Figures of Merit ........................................................................... 80
5.7 Scanning Variables ...................................................................... 80
5.8 Submission of New Models .......................................................... 81
5.9 Code Structure ........................................................................... 81
  5.9.1 Numerics modules ................................................................. 81
  5.9.2 Physics modules ................................................................... 81
  5.9.3 Engineering modules ............................................................ 82
  5.9.4 Costing module .................................................................... 82
  5.9.5 Other modules ...................................................................... 82

6 Utility Programs ........................................................................... 84

6.1 Executables ................................................................................. 84
  6.1.1 Turn output into input (write new_in.dat.py) ......................... 84
6.1.2 Randomly vary the starting point (run_process.py) .......................... 85
6.1.3 List PROCESS runs with comments (build_index.py) .......................... 86
6.1.4 test_process.py ........................................................................ 87
6.1.5 Output plotting: create data file (make_plot_dat.py) ......................... 88
6.1.6 Create a two-page summary (plot_proc.py) ..................................... 89
6.1.7 output_data.py ......................................................................... 89
6.1.8 plot_sweep.py .......................................................................... 90
6.1.9 Plot scan results (plot_mfile_sweep.py) ......................................... 90
6.1.10 Plot iteration variables and constraint residuals (diagnose_process.py) 90
6.1.11 fit_profile.py ........................................................................ 91
6.1.12 Step from one model to another (a_to_b.py) .................................. 91
6.1.13 N-Dimensional Scanner Utility .................................................... 93
6.1.14 Uncertainty Tools ..................................................................... 95
6.1.15 create_dicts.py ....................................................................... 99
6.1.16 Batch Jobs ............................................................................. 99
6.1.17 mfile_comparison.py ................................................................. 100
6.1.18 test_suite.py .......................................................................... 100
6.2 Python Libraries .......................................................................... 104
6.2.1 in_dat.py ............................................................................. 104
6.2.2 mfile.py .............................................................................. 105
6.2.3 process_funcs.py .................................................................... 106
6.2.4 process_config.py ................................................................. 107
6.2.5 process_dicts.py ................................................................. 109
6.2.6 a_to_b_config.py ................................................................. 109
6.2.7 proc_plot_func.py ................................................................. 110
6.2.8 diagnose_funcs.py ............................................................... 110
6.3 User Interface ............................................................................ 110

7 Code Management Tools ................................................................... 111
7.1 The Makefile ............................................................................... 111
7.1.1 Compilation ........................................................................ 111
7.1.2 Archiving utilities ............................................................... 112
7.1.3 Code documentation ............................................................ 112
7.2 Automatic Documentation ........................................................... 112
7.3 Code Updates and Release Procedure ........................................... 113
7.3.1 Initial access to the source code ............................................... 113
List of Figures

2.1 Flow diagram of PROCESS in non-optimisation mode ........................................ 12
2.2 Flow diagram of PROCESS in optimisation mode ............................................. 14
3.1 Machine build for D-shaped major components ............................................... 26
3.2 Machine build for elliptical-shaped major components ................................... 27
3.3 Machine build for a single-null device ............................................................ 29
3.4 Radiation power contributions ........................................................................ 34
3.5 Power balance within the core plasma .............................................................. 37
3.6 Schematic diagram of the cross-section of a superconducting TF coil inner leg .... 43
3.7 Coil and plasma current waveforms .................................................................. 46
3.8 Schematic diagram of the neutral beam access geometry .................................... 50
3.9 Power flow outside the fusion power plant core ............................................... 52
3.10 HELIAS 5-B Stellarator Power Plant Design ...................................................... 58
3.11 Stellarator island divertor configuration ........................................................... 61
6.1 PROCESS Test suite folder structure after a test run. The test_area folder is where the output of the test run is saved. .......................................................... 125
A.1 Illustration of Lagrange multiplier method ......................................................... 125
A.2 Flowchart of the VMCON optimisation solver .................................................... 128
List of Tables

2.1 List of iteration variables 1 to 55 .................................................. 17
2.2 List of iteration variables 56 onwards ........................................... 18
2.3 List of constraint equations 1–50 .................................................... 21
2.4 List of constraint equations 51 onwards ........................................ 22
2.5 List of figures of merit ................................................................. 23
2.6 List of scanning variables ......................................................... 24

3.1 List of available energy confinement scaling laws ............................. 36
3.2 List of available L-H power threshold scalings ................................. 38
3.3 PROCESS switches for spherical tokamaks ..................................... 55
3.4 Variables used in the hydrogen plant model .................................... 57
3.5 Parameters used in the Taylor-Ward availability model ....................... 67

5.1 Summary of numerics modules .................................................... 81
5.2 Summary of physics modules ....................................................... 82
5.3 Summary of engineering modules ............................................... 82
5.4 Summary of other modules ....................................................... 83

A.1 Summary of the description and meaning of the VMCON return parameters ifail ........................................ 129
Chapter 1

Introduction

1.1 Rationale

During the course of studies into a proposed fusion power plant, there may be times when questions of the following type arise:

- Are the machine’s physics and engineering parameters consistent with one another?
- Which machine of a given size and shape produces the cheapest electricity?
- What is the effect of a more optimistic limit on the maximum plasma density on the amount of auxiliary power required?

Questions such as these are extremely difficult to answer, since the large number of parameters involved are highly dependent on one another. Fortunately, computer programs have been written to address these issues, and PROCESS is one of them.

Suppose that an outline power plant design calls for a machine with a given size and shape, which will produce a certain net electric power. There may be a vast number of different conceptual machines that satisfy the problem as stated so far, and PROCESS can be used in “non-optimisation” mode to find one of these whose physics and engineering parameters are self-consistent. However, the machine found by PROCESS in this manner may not be possible to build in practice — the coils may be overstressed, for instance, or the plasma pressure may exceed the maximum possible value. PROCESS contains a large number of constraints to prevent the code from finding a machine with such problems, and running the code in so-called “optimisation” mode forces these constraints to be met. The number of possible conceptual machines is thus considerably reduced, and optimisation of the parameters with respect to (say) the cost of electricity will reduce this number to a minimum (possibly one).

Formally then, PROCESS is a systems code that calculates in a self-consistent manner the parameters of a fusion power plant with a specified performance, ensuring that its operating limits are not violated, and with the option to optimise a given function of these parameters.

1.2 History

PROCESS is derived from several earlier systems codes, but is largely based on the TETRA (Tokamak Engineering Test Reactor Analysis) code [1] and its descendant STORAC (Spherical TOrus Reactor Analysis Code) [2], which includes routines relevant to the spherical tokamak class of machines. These codes, and much of the original version of PROCESS itself, were written by personnel at Oak Ridge
National Laboratory in Tennessee, USA, with contributions from a number of other laboratories in the USA. In addition, many of the mathematical routines have been taken from a number of different well-established source libraries.

A great deal of effort was expended at Culham on the code’s arrival from ORNL in the early 1990s to upgrade and extend the code, including the addition of machines based on the stellerator, reversed field pinch and inertial confinement concepts.

PROCESS is being developed actively. This User Guide is updated in parallel with the code itself to ensure that the documentation remains consistent with the latest version of the code. It is to be hoped that it will be of assistance not only to users of PROCESS, but to anyone using PROCESS outputs or models based on them.

1.3 Sources of information

The output file OUT.DAT contains details of all the constraints and iteration variables selected, the input and output parameters, and which models have been chosen. It is essential to study the full output before using the results in any way.

Details of the physics, engineering and cost models are being published:

[19]: M. Kovari et al., PROCESS: a systems code for fusion power plants - Part 1: Physics
[20]: M. Kovari et al., PROCESS: a systems code for fusion power plants - Part 2: Engineering, in preparation
[21]: M. Kovari et al., The cost of a fusion power plant: extrapolation from ITER, in preparation

To view sample input and output files and a description of the variables, or to report bugs, request improvements, propose new models, or contact the PROCESS developers, please visit ccfe.ac.uk/powerplants.aspx
Chapter 2

Program Overview — The Fundamental Concepts

Fusion power plants are complex systems consisting of many non-linear interactions. One method that can be used to model this kind of system is to iterate a number of free parameters (the so-called iteration variables — see Section 2.4) in a controlled way so as to find a self-consistent set of device parameters that satisfy all of the system’s constraint equations — see Section 2.5. PROCESS is organised in a standard equation solver format to enable this task to be performed efficiently. The physics and engineering routines together serve as a function evaluator, providing the information used in the solution of the constraints. The numerical modules present in PROCESS perform the iteration required, and also incorporate the option to maximise or minimise a given figure of merit — see Section 2.6.

2.1 Equation Solvers

PROCESS contains a constrained non-linear optimiser, and a constrained non-linear equation solver, which reflect the two modes of operation available. Each of these has its own uses, as is now discussed.

2.1.1 Non-optimisation mode

The first of the two equation solvers present in PROCESS is the non-optimisation package HYBRD [3, 4]. Formally, HYBRD finds a zero of a system of \(N\) non-linear functions in \(N\) variables. This means simply that \(N\) variables (power plant parameters) are iterated by PROCESS in such a way as to solve a set of \(N\) equations (physics or engineering laws), i.e. a set of self-consistent power plant parameters is found. This is useful for performing benchmark comparisons, when the device size is kept fixed, and one only wishes to find calculated stresses, beta values, fusion powers, etc. A flow diagram of PROCESS in non-optimisation mode is shown in Figure 2.1.

2.1.2 Optimisation mode

Sometimes one wants to find an optimal machine that is both consistent with the physics and engineering constraints but also minimises or maximises a certain figure of merit. This requires running PROCESS in optimisation mode. For these applications PROCESS uses the routine VMCON [5] based on a variable metric method for constrained optimisation by Powell [6]. It finds a stationary point of an objective function/figure of merit consistent with a set of equality and inequality constraints.
Figure 2.1: Flow diagram of PROCESS in non-optimisation mode.
It is designed to solve the necessary but not sufficient conditions of a constrained optimum, hence, the solution does not have to be a global optimum (see Section A.2).

The detailed algorithm is explained in Appendix A and is based on the Lagrange method (see Section A.2) that combines both the objective function and the constraints using Lagrange multipliers. It applies a sequential quadratic programming approach (Section A.3) in which a series of subproblems is solved that constitute a local quadratic expansion of the Lagrangian (Section A.5). The solution of the quadratic subproblem is the direction of a line search along which a one dimensional optimisation is performed (Section A.6). This line search was introduced by Powell to assure convergence from bad starting parameters. Within PROCESS we have modified the line search of the original VMCON routine to (try to) assure convergence, even for slightly inconsistent input functions.

The algorithm uses a quasi-Newtonian approach which only requires continuous first derivatives of these functions. While the first derivatives are evaluated using a finite difference approximation, the second derivatives are estimated using a variant of the Broyden-Fletcher-Goldfarb-Shanno update (see Section A.7).

The convergence criterion of the solver as well as the detailed interpretation of the various error codes are explained in Section A.4. A flow diagram of PROCESS in optimisation mode is shown in Figure 2.2.
Chapter 2  Program Overview — The Fundamental Concepts

initialise variables

input from file

define free parameters (iteration variables) and their bounds

define constraints

define figure-of-merit

evaluate physics, engineering and cost functions

apply consistency equations and limit equations

self-consistent?

no

F-o-M minimised?

no

write output

iterate free parameters

yes

yes

Figure 2.2: Conceptual flow diagram of PROCESS in optimisation mode. Please note that this is a simplistic interpretation of the actual sequence of operations, to outline the difference between non-optimisation and optimisation modes. For a more detailed (and correct!) VMCON flow diagram, please see Figure A.2.
2.1.3 Scans

It is often useful to be able to scan through a range of values of a given parameter to see what effect this has on the machine as a whole. Sensitivity studies of this kind can be achieved very easily using PROCESS. Scans are carried out in optimisation mode, whereby the code initially performs a run using the parameters specified in the input file, and then a series of runs using the parameters produced at the end of the previous iteration. The value of the quantity being scanned is specified at every stage — see Section 2.7. This method ensures that a smooth variation in the machine parameters is achieved. A tool to carry out scans in any number of parameters is under development.

2.2 The Variable Descriptor File

The variable descriptor file vardes.html is an invaluable resource for the user of PROCESS. It acts as a dictionary/reference manual for the code’s variables, and contains the following information about each:

- name
- dimensions (of arrays)
- default value(s)
- description, including physical units if relevant
- for switches/flags, the meanings of all allowed values
- iteration variable number, if relevant
- corresponding constraint equation, if relevant

In addition, global code parameters are labelled FIX. These can only be changed by editing the relevant source file, but this should not be carried out unless it is absolutely necessary.

All the variables that are shown with a default value are available to be changed by the user using the input file (Section 4.4), except for those which are labelled FIX. Variables not shown with a default value are calculated by the code from a combination of other parameters, and so it would be meaningless to initialise them. Obviously, these variables cannot be changed using the input file.

2.3 Input Parameters

Input parameters make up a large proportion of the variables listed in the variable descriptor file. They comprise all those variables that, once set in the input file (or set to a default value), do not change throughout a PROCESS run. For example, the variable fdeut is input, and cannot change during a run, appearing in vardes.html as follows: fdeut /0.5/ : deuterium fuel fraction.

2.4 Iteration Variables

Variables that are adjusted by PROCESS in order to satisfy the constraints and optimise the figure of merit are referred to as iteration variables. Successive calls are made to the physics and engineering
routines, with slightly different values for the iteration variables on each call, and the equation solver determines the effect on the output due to these small changes to the input (see Figures 2.1 and 2.2). Tables 2.1 and 2.2 list the iteration variables available in PROCESS. Any one or more of these variables may be selected.

Clearly, the equation solvers need at least as many variables to iterate as there are equations to solve, i.e. \( n_{\text{var}} \geq n_{\text{eqns}} \). If the run is a non-optimising case, then \( n_{\text{eqns}} \) variables are iterated — the values of the remaining \( (n_{\text{var}}-n_{\text{eqns}}) \) variables are left alone. If the run is an optimising case, then all the active iteration variables are adjusted so as to find the minimum (or maximum) value of a parameter (the figure of merit) in the \( n_{\text{var}} \)-dimensional space of the problem.

In optimisation mode, all the iteration variables are constrained to lie between lower and upper bounds. For instance, the plasma electron density is, by default, confined to lie between the values \( 10^{19} \text{ m}^{-3} \) and \( 10^{21} \text{ m}^{-3} \). Of course, it can also be constrained to lie below the calculated density limit, if constraint equation 5 is activated and the f-value \( f_{\text{dene}} \) (iteration variable no. 9) is bounded by the values 0 and 1.

It is important to remember that iteration variables must never be initialised to zero. The code will not be able to adjust the variable’s value if this is done, and it will stop with an error message.

For example, the major radius is available as an iteration variable, and appears in the variable description file as \texttt{rmajor /8.14/ : plasma major radius (m) (iteration variable 3)}. If it is selected as an iteration variable, it will be adjusted by the code. The value input by the user (or the default, if no value is specified), will be used as the starting value.
<table>
<thead>
<tr>
<th>no.</th>
<th>variable name</th>
<th>description</th>
<th>icc</th>
<th>lower bound</th>
<th>upper bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>aspect</td>
<td>plasma aspect ratio</td>
<td>1.100D0</td>
<td>10.00D0</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>bt</td>
<td>toroidal field on axis</td>
<td>0.010D0</td>
<td>100.0I</td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>rmajor</td>
<td>plasma major radius</td>
<td>0.100D0</td>
<td>10.00D0</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>te</td>
<td>electron temperature</td>
<td>5.000D0</td>
<td>500.0D0</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>beta</td>
<td>plasma beta</td>
<td>0.001D0</td>
<td>1.000D0</td>
<td></td>
</tr>
<tr>
<td>6</td>
<td>dene</td>
<td>electron density</td>
<td>1.00D19</td>
<td>1.00D21</td>
<td></td>
</tr>
<tr>
<td>7</td>
<td>rnbm</td>
<td>hot beam density / electron density</td>
<td>1.00D6-1</td>
<td>1.00D0</td>
<td></td>
</tr>
<tr>
<td>8</td>
<td>fbet</td>
<td>f-value for β limit equation</td>
<td>6</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>9</td>
<td>fden</td>
<td>f-value for density limit equation</td>
<td>5</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>10</td>
<td>hfact</td>
<td>confinement time H-factor</td>
<td>0.100D0</td>
<td>3.000D0</td>
<td></td>
</tr>
<tr>
<td>11</td>
<td>pheat</td>
<td>heating power not used for current drive</td>
<td>0.001D0</td>
<td>1.000D3</td>
<td></td>
</tr>
<tr>
<td>12</td>
<td>oacdcp</td>
<td>overall current density in TF coil inboard leg</td>
<td>1.000D5</td>
<td>1.500D8</td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>tfcth</td>
<td>TF coil inboard leg thickness</td>
<td>1.000D0</td>
<td>5.000D0</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>fwall</td>
<td>f-value for wall load limit equation</td>
<td>8</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>15</td>
<td>fvs</td>
<td>f-value for volt second limit equation</td>
<td>12</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>16</td>
<td>ocnth</td>
<td>central solenoid thickness</td>
<td>0.001D0</td>
<td>1.000D2</td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>tdwell</td>
<td>dwell time</td>
<td>0.100D0</td>
<td>1.000D8</td>
<td></td>
</tr>
<tr>
<td>18</td>
<td>q</td>
<td>edge safety factor</td>
<td>2.000D0</td>
<td>100.0D0</td>
<td></td>
</tr>
<tr>
<td>19</td>
<td>enbeam</td>
<td>neutral beam energy</td>
<td>1.000D0</td>
<td>1.000D6</td>
<td></td>
</tr>
<tr>
<td>20</td>
<td>tcpav</td>
<td>average (resistive) TF coil temperature</td>
<td>40.0D0</td>
<td>1.000D3</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>fburn</td>
<td>f-value for burn time limit equation</td>
<td>13</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>22</td>
<td>tbrnm</td>
<td>minimum burn time</td>
<td>0.001D0</td>
<td>1.000D6</td>
<td></td>
</tr>
<tr>
<td>23</td>
<td>fcoolcp</td>
<td>coolant fraction of resistive TF coil</td>
<td>0.100D0</td>
<td>0.500D0</td>
<td></td>
</tr>
<tr>
<td>24</td>
<td>cdtfleg</td>
<td>TF coil leg overall current density</td>
<td>1.000D4</td>
<td>1.000D8</td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>fnetel</td>
<td>f-value for net electric power limit equation</td>
<td>16</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>26</td>
<td>ffuspow</td>
<td>f-value for fusion power limit equation</td>
<td>9</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>27</td>
<td>fhldiv</td>
<td>f-value for divertor heat load limit equation</td>
<td>18</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>28</td>
<td>fradpwr</td>
<td>f-value for radiation power limit equation</td>
<td>17</td>
<td>0.001D0</td>
<td>0.990D0</td>
</tr>
<tr>
<td>29</td>
<td>bore</td>
<td>machine bore</td>
<td>0.100D0</td>
<td>10.00D0</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>fsva</td>
<td>f-value for MVA limit equation</td>
<td>19</td>
<td>0.010D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>31</td>
<td>gapomin</td>
<td>minimum gap between outboard vacuum vessel and TF coil</td>
<td>0.001D0</td>
<td>1.000D0</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>frminor</td>
<td>f-value for minor radius limit equation</td>
<td>21</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>33</td>
<td>fportsz</td>
<td>f-value for beam tangency radius limit equation</td>
<td>20</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>34</td>
<td>fdivcol</td>
<td>f-value for divertor collisionality limit equation</td>
<td>22</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>35</td>
<td>fpeakk</td>
<td>f-value for peak toroidal field limit equation</td>
<td>25</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>36</td>
<td>fbetatry</td>
<td>f-value for beta limit equation</td>
<td>24</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>37</td>
<td>coheof</td>
<td>central solenoid current density at end of flat-top</td>
<td>1.000D5</td>
<td>1.000D8</td>
<td></td>
</tr>
<tr>
<td>38</td>
<td>fjohc</td>
<td>f-value for central solenoid current at EOF limit equation</td>
<td>26</td>
<td>0.010D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>39</td>
<td>fjohc0</td>
<td>f-value for central solenoid current at BOP limit equation</td>
<td>27</td>
<td>0.010D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>40</td>
<td>fgamcd</td>
<td>f-value for current drive gamma limit equation</td>
<td>37</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>41</td>
<td>fcohbop</td>
<td>central solenoid current density ratio BOP/EOF</td>
<td>0.001D0</td>
<td>1.000D0</td>
<td></td>
</tr>
<tr>
<td>42</td>
<td>gapoh</td>
<td>gap between central solenoid and TF coil</td>
<td>0.001D0</td>
<td>1.000D0</td>
<td></td>
</tr>
<tr>
<td>43</td>
<td>cfe0</td>
<td>seeded high-Z impurity fraction</td>
<td>1.000D6</td>
<td>3.000D3</td>
<td></td>
</tr>
<tr>
<td>44</td>
<td>fvsbrnmi</td>
<td>fraction of plasma current produced by non-inductive means</td>
<td>0.001D0</td>
<td>1.000D0</td>
<td></td>
</tr>
<tr>
<td>45</td>
<td>fqval</td>
<td>f-value for fusion gain limit equation</td>
<td>28</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>46</td>
<td>fpinj</td>
<td>f-value for injection power limit equation</td>
<td>30</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>47</td>
<td>feffcd</td>
<td>current drive efficiency multiplier</td>
<td>0.001D0</td>
<td>1.000D0</td>
<td></td>
</tr>
<tr>
<td>48</td>
<td>fstrcase</td>
<td>f-value for TF coil case stress limit equation</td>
<td>31</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>49</td>
<td>fstrcond</td>
<td>f-value for TF coil conduit stress limit equation</td>
<td>32</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>50</td>
<td>fiooioc</td>
<td>f-value for TF coil operational current limit equation</td>
<td>33</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>51</td>
<td>fvdump</td>
<td>f-value for TF coil dump voltage limit equation</td>
<td>34</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>52</td>
<td>vdalw</td>
<td>allowable TF coil dump voltage</td>
<td>0.001D0</td>
<td>1.000D0</td>
<td></td>
</tr>
<tr>
<td>53</td>
<td>fjprot</td>
<td>f-value for TF coil current protection limit equation</td>
<td>35</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>54</td>
<td>ftmgntf</td>
<td>f-value for TF coil temperature margin limit equation</td>
<td>36</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>55</td>
<td></td>
<td>(OBSOLETE)</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

Table 2.1: Iteration variables 1 to 55 present in PROCESS. The f-values correspond to the given constraint equations (see Table 2.3). The other iteration variables are shown in Table 2.2.
<table>
<thead>
<tr>
<th>no.</th>
<th>variable name</th>
<th>description</th>
<th>icc</th>
<th>lower bound</th>
<th>upper bound</th>
</tr>
</thead>
<tbody>
<tr>
<td>56</td>
<td>tdmptf</td>
<td>dump time for TF coil</td>
<td>10.00D0</td>
<td>1.000D0</td>
<td></td>
</tr>
<tr>
<td>57</td>
<td>thccas</td>
<td>TF coil external case thickness</td>
<td>0.050D0</td>
<td>1.000D0</td>
<td></td>
</tr>
<tr>
<td>58</td>
<td>thvccas</td>
<td>TF coil conduit case thickness</td>
<td>0.001D0</td>
<td>1.000D0</td>
<td></td>
</tr>
<tr>
<td>59</td>
<td>fcutfsu</td>
<td>copper fraction of cable conductor</td>
<td>0.001D0</td>
<td>1.000D0</td>
<td></td>
</tr>
<tr>
<td>60</td>
<td>cpttf</td>
<td>current per turn in the TF coils</td>
<td>0.001D0</td>
<td>4.000D4</td>
<td></td>
</tr>
<tr>
<td>61</td>
<td>gaps</td>
<td>gap between vacuum vessel and inboard TF coil</td>
<td>0.001D0</td>
<td>10.00D0</td>
<td></td>
</tr>
<tr>
<td>62</td>
<td>fdtemp</td>
<td>f-value for 1st wall coolant temperature rise limit equation</td>
<td>38</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>63</td>
<td>ftpeak</td>
<td>f-value for 1st wall peak temperature limit equation</td>
<td>39</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>64</td>
<td>fnxmn</td>
<td>f-value for minimum auxiliary power limit equation</td>
<td>40</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>65</td>
<td>tohs</td>
<td>central solenoid current ramp-up time</td>
<td>0.100D0</td>
<td>1.000D0</td>
<td></td>
</tr>
<tr>
<td>66</td>
<td>ftohs</td>
<td>f-value for central solenoid current ramp-up time limit equation</td>
<td>41</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>67</td>
<td>ftcycl</td>
<td>f-value for minimum cycle time limit equation</td>
<td>42</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>68</td>
<td>ftemp</td>
<td>f-value for maximum centrepot temperature limit equation</td>
<td>44</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>69</td>
<td>rcool</td>
<td>average radius of centrepot coolant channel</td>
<td>0.001D0</td>
<td>0.010D0</td>
<td></td>
</tr>
<tr>
<td>70</td>
<td>vcool</td>
<td>maximum centrepot coolant flow speed at midplane</td>
<td>1.000D0</td>
<td>1.000D0</td>
<td></td>
</tr>
<tr>
<td>71</td>
<td>fq</td>
<td>f-value for minimum edge safety factor limit equation</td>
<td>45</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>72</td>
<td>fipir</td>
<td>f-value for maximum I_f/I_rad limit equation</td>
<td>46</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>73</td>
<td>scrapli</td>
<td>inboard gap between plasma and first wall</td>
<td>0.001D0</td>
<td>10.00D0</td>
<td></td>
</tr>
<tr>
<td>74</td>
<td>scraplo</td>
<td>outboard gap between plasma and first wall</td>
<td>0.001D0</td>
<td>10.00D0</td>
<td></td>
</tr>
<tr>
<td>75</td>
<td>tfootf</td>
<td>ratio of TF coil outboard/inboard leg thickness</td>
<td>0.200D0</td>
<td>5.000D0</td>
<td></td>
</tr>
<tr>
<td>76</td>
<td>frfptf</td>
<td>f-value for TF coil toroidal thickness limit equation</td>
<td>47</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>77</td>
<td>tftort</td>
<td>TF coil toroidal thickness (use for RFPs only)</td>
<td>0.050D0</td>
<td>4.000D0</td>
<td></td>
</tr>
<tr>
<td>78</td>
<td>rpth</td>
<td>RFP pinch parameter, Θ</td>
<td>0.010D0</td>
<td>1.800D0</td>
<td></td>
</tr>
<tr>
<td>79</td>
<td>fbetap</td>
<td>f-value for poloidal beta limit equation</td>
<td>48</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>80</td>
<td>frfpe</td>
<td>f-value for RFP reversal parameter limit equation</td>
<td>49</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>81</td>
<td>edrive</td>
<td>IFE driver energy</td>
<td>1.000D5</td>
<td>5.000D7</td>
<td></td>
</tr>
<tr>
<td>82</td>
<td>drveff</td>
<td>IFE driver wall plug to target efficiency</td>
<td>0.010D0</td>
<td>1.000D0</td>
<td></td>
</tr>
<tr>
<td>83</td>
<td>tgain</td>
<td>IFE target gain</td>
<td>1.000D0</td>
<td>500.0D0</td>
<td></td>
</tr>
<tr>
<td>84</td>
<td>chrad</td>
<td>radius of IFE chamber</td>
<td>0.010D0</td>
<td>20.00D0</td>
<td></td>
</tr>
<tr>
<td>85</td>
<td>pdrive</td>
<td>IFE driver power reaching target</td>
<td>1.000D6</td>
<td>2.000D8</td>
<td></td>
</tr>
<tr>
<td>86</td>
<td>frmax</td>
<td>f-value for maximum IFE repetition rate equation</td>
<td>50</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>87</td>
<td>helelmw</td>
<td>electrical power required for hydrogen production</td>
<td>1.000D0</td>
<td>4.000D3</td>
<td></td>
</tr>
<tr>
<td>88</td>
<td>hthermmw</td>
<td>thermal power required for hydrogen production</td>
<td>1.000D0</td>
<td>4.000D3</td>
<td></td>
</tr>
<tr>
<td>89</td>
<td>ftr</td>
<td>f-value for tritium breeding ratio limit equation</td>
<td>52</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>90</td>
<td>blbuith</td>
<td>inboard blanket breeding unit thickness</td>
<td>0.001D0</td>
<td>2.000D0</td>
<td></td>
</tr>
<tr>
<td>91</td>
<td>blbuoth</td>
<td>outboard blanket breeding unit thickness</td>
<td>0.001D0</td>
<td>2.000D0</td>
<td></td>
</tr>
<tr>
<td>92</td>
<td>fflutf</td>
<td>f-value for fast neutron fluence on TF coil equation</td>
<td>53</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>93</td>
<td>sldith</td>
<td>inboard shield thickness</td>
<td>0.001D0</td>
<td>10.00D0</td>
<td></td>
</tr>
<tr>
<td>94</td>
<td>sldoth</td>
<td>outboard shield thickness</td>
<td>0.001D0</td>
<td>10.00D0</td>
<td></td>
</tr>
<tr>
<td>95</td>
<td>fptfnuc</td>
<td>f-value for TF coil nuclear heating limit equation</td>
<td>54</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>96</td>
<td>fvvhe</td>
<td>f-value for vessel He concentration limit equation</td>
<td>55</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>97</td>
<td>fpepr</td>
<td>f-value for P_separatrix/R_major limit equation</td>
<td>56</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>98</td>
<td>li6enrich</td>
<td>lithium-6 enrichment percentage (blktmodel=1)</td>
<td>7.400D0</td>
<td>100.0D0</td>
<td></td>
</tr>
<tr>
<td>99</td>
<td>(OBSOLET)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>100</td>
<td>(OBSOLET)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>101</td>
<td>prp</td>
<td>ratio of TF coil radial plate area to winding pack area</td>
<td>1.000D6</td>
<td>0.010D0</td>
<td></td>
</tr>
<tr>
<td>102</td>
<td>fimpvar</td>
<td>impurity fraction of element impvar</td>
<td>1.000D6</td>
<td>0.010D0</td>
<td></td>
</tr>
<tr>
<td>103</td>
<td>fithresh</td>
<td>f-value for L-H power threshold limit equation</td>
<td>15</td>
<td>1.000D0</td>
<td>1.000D6</td>
</tr>
<tr>
<td>104</td>
<td>fcr</td>
<td>f-value for conducting shell radius limit equation</td>
<td>23</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>105</td>
<td>fnbshinef</td>
<td>f-value for NBI shine-through fraction limit equation</td>
<td>59</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>106</td>
<td>ftnf</td>
<td>f-value for CS coil temperature margin limit equation</td>
<td>60</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>107</td>
<td>favail</td>
<td>f-value for minimum availability limit equation</td>
<td>61</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>108</td>
<td>breeder_f</td>
<td>breeder volume ratio: Li4SiO4/(Be12Ti+Li4SiO4)</td>
<td>0.060D0</td>
<td>1.000D0</td>
<td></td>
</tr>
<tr>
<td>109</td>
<td>ralne</td>
<td>thermal alpha density / electron density</td>
<td>0.050D0</td>
<td>0.150D0</td>
<td></td>
</tr>
<tr>
<td>110</td>
<td>ftaulimit</td>
<td>f-value for limit on ratio of alpha particle to energy confinement times</td>
<td>62</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>111</td>
<td>fniterpump</td>
<td>f-value for constraint that number of pumps i tfno</td>
<td>63</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>112</td>
<td>fzeffmax</td>
<td>f-value for Zeff limit equation</td>
<td>64</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>113</td>
<td>ftaucq</td>
<td>f-value for quench time limit equation</td>
<td>65</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
<tr>
<td>114</td>
<td>fchannel_length</td>
<td>Length of a single first wall channel</td>
<td>39</td>
<td>0.001D0</td>
<td>1.000D3</td>
</tr>
<tr>
<td>115</td>
<td>fpoloidalpower</td>
<td>f-value for max rate of change of energy in poloidal field</td>
<td>66</td>
<td>0.001D0</td>
<td>1.000D0</td>
</tr>
</tbody>
</table>

Table 2.2: Iteration variables 56 onwards. The f-values correspond to the given constraint equations (see Table 2.3). The other iteration variables are shown in Table 2.1.
2.5 Constraint Equations

Any computer program naturally contains myriads of equations. The built-in equation solvers within PROCESS act on a special class, known as constraint equations, all of which are formulated in routine constraint_eqns in source file constraint_equations.f90. Table 2.3 summarises the constraint equations available in PROCESS. These can be split into two types — (1) consistency equations, that enforce consistency between the physics and engineering parameters, and (2) limit equations, that enforce various parameters to lie within their allowed limits. The neqns constraint equations that the user chooses for a given run are activated by including the equation numbers in the first neqns elements of array icc.

2.5.1 Consistency equations

Consistency equations are equalities that ensure that the machine produced by PROCESS is self-consistent. This means, therefore, that many of these constraint equations should always be used, namely equations 1, 2, 10 and 11 (see Table 2.3). Equation 7 should also be activated if neutral beam injection is used. The other consistency equations can be activated if required.

A typical consistency equation ensures that two functions $g$ and $h$ are equal:

\[
g(x, y, z, \ldots) = h(x, y, z, \ldots)
\]

\[
c_i = 1 - \frac{g}{h}
\]

The equation solvers VMCON and HYBRD need the constraint equations $c_i$ to be given in the form shown, since they adjust the iteration variables so as to obtain $c_i = 0$, thereby ensuring that $g = h$.

2.5.2 Limit equations

The limit equations are inequalities that ensure that various physics or engineering limits are not exceeded. Each of these equations has an associated f-value, which allow them to be coded as equalities. The f-values are used as follows.

In general, limit equations have the form

\[
c_i = 1 - f \cdot \frac{h_{\text{max}}}{h}
\]

where $f$ is the f-value. In optimisation mode, all iteration variables have prescribed lower and upper bounds. If $f$ is chosen to be an iteration variable and is given a lower bound of zero and an upper bound of one, then the limit equation does indeed constrain the calculated quantity to lie between zero and its maximum allowable value, as required.

As with the consistency equations, the general form of the limit equations is

\[
c_i = 1 - f \cdot \frac{h_{\text{max}}}{h}
\]

where $h_{\text{max}}$ is the maximum allowed value of the quantity $h$.

Sometimes, the limit equation and f-value are used to ensure that quantity $h$ is larger than its minimum value $h_{\text{min}}$. In this case, $0 \leq f \leq 1$ (as before), but the equation takes the form

\[
c_i = 1 - f \cdot \frac{h}{h_{\text{min}}}
\]
By fixing the f-value (i.e. not including it in the $ixc$ array), the limit equations can be used as equality constraints. For example, to set the net electric power to a certain value, the following should be carried out:

1. Activate constraint 16 (net electric power lower limit) by including it in the first $neqns$ elements of array $icc$
2. Set the corresponding f-value $fpnetel = 1.0D0$
3. Ensure that $fpnetel$ (iteration variable no. 25) *IS NOT* selected as an iteration variable.
4. Set $pnetelin$ to the required net electric power.

Limit equations are not restricted to optimisation mode. In non-optimisation mode, the iteration variables are not bounded, but the f-values can still be used to provide information about how calculated values compare with limiting values, without having to change the characteristics of the device being benchmarked to find a solution.

It is for this reason that all the constraint equations used in PROCESS are formulated as equalities, despite the fact that equation solver VMCON can solve for inequalities as well. The use of f-values precludes this need, and allows the non-optimising equation solver HYBRD to use the same constraint equations.
<table>
<thead>
<tr>
<th>icc no.</th>
<th>description</th>
<th>type</th>
<th>corresponding icc variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>plasma beta consistency</td>
<td>C</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>global power balance</td>
<td>C</td>
<td>10,1,2,3,4,6,11</td>
</tr>
<tr>
<td>3</td>
<td>ion power balance (DEPRECATED - DO NOT USE)</td>
<td>C</td>
<td>10,1,2,3,4,6,11</td>
</tr>
<tr>
<td>4</td>
<td>electron power balance (DEPRECATED - DO NOT USE)</td>
<td>C</td>
<td>10,1,2,3,4,6,11</td>
</tr>
<tr>
<td>5</td>
<td>density upper limit</td>
<td>L</td>
<td>9,1,2,3,4,5,6</td>
</tr>
<tr>
<td>6</td>
<td>epsilon-beta poloidal upper limit</td>
<td>L</td>
<td>8,1,2,3,4,6</td>
</tr>
<tr>
<td>7</td>
<td>beam ion density (NBI)</td>
<td>C</td>
<td>7</td>
</tr>
<tr>
<td>8</td>
<td>neutron wall load upper limit</td>
<td>L</td>
<td>14,1,2,3,4,6</td>
</tr>
<tr>
<td>9</td>
<td>fusion power upper limit</td>
<td>L</td>
<td>26,1,2,3,4,6</td>
</tr>
<tr>
<td>10</td>
<td>toroidal field 1/R consistency</td>
<td>C</td>
<td>12,1,2,3,13</td>
</tr>
<tr>
<td>11</td>
<td>radial build consistency</td>
<td>C</td>
<td>3,1,13,16,29,42,61</td>
</tr>
<tr>
<td>12</td>
<td>volt second capability lower limit</td>
<td>L</td>
<td>15,1,2,3</td>
</tr>
<tr>
<td>13</td>
<td>burn time lower limit (PULSE)</td>
<td>L</td>
<td>21,1,16,17,22,92,42,44,61</td>
</tr>
<tr>
<td>14</td>
<td>neutral beam decay lengths = tbeamin consistency (NBI)</td>
<td>L</td>
<td>25,1,2,3,6</td>
</tr>
<tr>
<td>15</td>
<td>L-H power threshold limit</td>
<td>L</td>
<td>103</td>
</tr>
<tr>
<td>16</td>
<td>net electric power lower limit</td>
<td>L</td>
<td>28</td>
</tr>
<tr>
<td>17</td>
<td>radiation fraction upper limit</td>
<td>L</td>
<td>27</td>
</tr>
<tr>
<td>18</td>
<td>divertor heat load upper limit</td>
<td>L</td>
<td>30</td>
</tr>
<tr>
<td>19</td>
<td>MVA upper limit</td>
<td>L</td>
<td>33,31,3,13</td>
</tr>
<tr>
<td>20</td>
<td>neutral beam tangency radius upper limit (NBI)</td>
<td>L</td>
<td>32</td>
</tr>
<tr>
<td>21</td>
<td>minor radius lower limit</td>
<td>L</td>
<td>34,43</td>
</tr>
<tr>
<td>22</td>
<td>divertor collisionality upper limit</td>
<td>L</td>
<td>104,1,74</td>
</tr>
<tr>
<td>23</td>
<td>conducting shell to plasma minor radius ratio upper limit</td>
<td>L</td>
<td>36,1,2,3,4,6,18</td>
</tr>
<tr>
<td>24</td>
<td>Beta upper limit (also beta limit in stellarators)</td>
<td>L</td>
<td>35,3,13,20</td>
</tr>
<tr>
<td>25</td>
<td>peak toroidal field upper limit</td>
<td>L</td>
<td>38,37,41,12</td>
</tr>
<tr>
<td>26</td>
<td>central solenoid current density at End of Flat-top upper limit</td>
<td>L</td>
<td>39,37,41,12</td>
</tr>
<tr>
<td>27</td>
<td>central solenoid current density at Beginning of Pulse upper limit</td>
<td>L</td>
<td>45,47,40</td>
</tr>
<tr>
<td>28</td>
<td>fusion gain Q lower limit</td>
<td>L</td>
<td>46,47,11</td>
</tr>
<tr>
<td>29</td>
<td>inboard radial build = specified value</td>
<td>C</td>
<td>48,56,57,58,59,60,24</td>
</tr>
<tr>
<td>30</td>
<td>injection power upper limit</td>
<td>L</td>
<td>49,56,57,58,59,60,24</td>
</tr>
<tr>
<td>31</td>
<td>TF coil case stress upper limit (SCTF)</td>
<td>L</td>
<td>50,56,57,58,59,60,24</td>
</tr>
<tr>
<td>32</td>
<td>TF coil conduit stress upper limit (SCTF)</td>
<td>L</td>
<td>51,52,56,57,58,59,60,24</td>
</tr>
<tr>
<td>33</td>
<td>TF coil Ioperational/Jcritical upper limit (SCTF)</td>
<td>L</td>
<td>53,56,57,58,59,60,24</td>
</tr>
<tr>
<td>34</td>
<td>TF coil dump voltage upper limit (SCTF)</td>
<td>L</td>
<td>54,55,56,57,58,59,60,24</td>
</tr>
<tr>
<td>35</td>
<td>TF coil Jwinding pack/Jprotection upper limit (SCTF)</td>
<td>L</td>
<td>55,56,57,58,59,60,24</td>
</tr>
<tr>
<td>36</td>
<td>TF coil temperature margin lower limit (SCTF)</td>
<td>L</td>
<td>56,57,58,59,60,24</td>
</tr>
<tr>
<td>37</td>
<td>current drive gamma upper limit</td>
<td>L</td>
<td>60,47</td>
</tr>
<tr>
<td>38</td>
<td>first wall coolant temperature rise upper limit (PULSE)</td>
<td>L</td>
<td>62</td>
</tr>
<tr>
<td>39</td>
<td>first wall peak temperature upper limit (PULSE)</td>
<td>L</td>
<td>63</td>
</tr>
<tr>
<td>40</td>
<td>injection power lower limit (PULSE)</td>
<td>L</td>
<td>64</td>
</tr>
<tr>
<td>41</td>
<td>central solenoid current ramp-up time lower limit (PULSE)</td>
<td>L</td>
<td>66,65</td>
</tr>
<tr>
<td>42</td>
<td>cycle time lower limit (PULSE)</td>
<td>L</td>
<td>67,65,17</td>
</tr>
<tr>
<td>43</td>
<td>average centrepost temperature (ST)</td>
<td>C</td>
<td>69,70,13</td>
</tr>
<tr>
<td>44</td>
<td>peak centrepost temperature upper limit (ST)</td>
<td>L</td>
<td>68,69,70</td>
</tr>
<tr>
<td>45</td>
<td>edge safety factor lower limit (ST)</td>
<td>L</td>
<td>71,1,2,3</td>
</tr>
<tr>
<td>46</td>
<td>Ip/Irod upper limit (ST)</td>
<td>L</td>
<td>72,2,60</td>
</tr>
<tr>
<td>47</td>
<td>TF coil toroidal thickness upper limit (RFP)</td>
<td>L</td>
<td>76,77,13,3</td>
</tr>
<tr>
<td>48</td>
<td>poloidal beta upper limit</td>
<td>L</td>
<td>79,2,3,18</td>
</tr>
<tr>
<td>49</td>
<td>reversal parameter &lt; 0 (RFP)</td>
<td>L</td>
<td>80,78,3,1</td>
</tr>
<tr>
<td>50</td>
<td>IFE repetition rate upper limit (IFE)</td>
<td>L</td>
<td>86</td>
</tr>
</tbody>
</table>

Table 2.3: Summary of the first 50 constraint equations present in PROCESS. Consistency equations are marked C, limit equations are marked L. Some (non-exhaustive) iteration variable numbers (see Tables 2.1 and 2.2) that directly affect the associated constraint equations are given, the one listed first being the most relevant.
<table>
<thead>
<tr>
<th>icc no.</th>
<th>description</th>
<th>type</th>
<th>corresponding icc variables</th>
</tr>
</thead>
<tbody>
<tr>
<td>51</td>
<td>startup volt-seconds consistency (PULSE)</td>
<td>C</td>
<td>16,29,3,1</td>
</tr>
<tr>
<td>52</td>
<td>tritium breeding ratio lower limit</td>
<td>L</td>
<td>89,90,91</td>
</tr>
<tr>
<td>53</td>
<td>peak neutron fluence on TF coil upper limit</td>
<td>L</td>
<td>92,93,94</td>
</tr>
<tr>
<td>54</td>
<td>peak TF coil nuclear heating upper limit</td>
<td>L</td>
<td>95,93,94</td>
</tr>
<tr>
<td>55</td>
<td>final He concentration in vacuum vessel upper limit</td>
<td>L</td>
<td>96,93,94</td>
</tr>
<tr>
<td>56</td>
<td>( P_{\text{separatrix}}/R_{\text{major}} ) upper limit</td>
<td>L</td>
<td>97,1,3,102</td>
</tr>
<tr>
<td>57</td>
<td>(OBSOLETE) TF coil outer leg toroidal thickness lower limit (SCTF)</td>
<td>L</td>
<td>99,29,13</td>
</tr>
<tr>
<td>58</td>
<td>(OBSOLETE) TF coil outer leg radial thickness lower limit (SCTF)</td>
<td>L</td>
<td>100,13</td>
</tr>
<tr>
<td>59</td>
<td>Neutral beam shine-through fraction upper limit (NBI)</td>
<td>L</td>
<td>105,6,19,4</td>
</tr>
<tr>
<td>60</td>
<td>Central solenoid temperature margin lower limit (SCTF)</td>
<td>L</td>
<td>106</td>
</tr>
<tr>
<td>61</td>
<td>Minimum availability value limit</td>
<td>L</td>
<td>107</td>
</tr>
<tr>
<td>62</td>
<td>Lower limit on the ratio of alpha particle to energy confinement times</td>
<td>L</td>
<td>110</td>
</tr>
<tr>
<td>63</td>
<td>Number of vacuum pumps (vacuum_model=simple)</td>
<td>L</td>
<td>111</td>
</tr>
<tr>
<td>64</td>
<td>Limit on ( Z_{\text{eff}} \leq Z_{\text{effmax}} )</td>
<td>L</td>
<td>112</td>
</tr>
<tr>
<td>65</td>
<td>TF dump time greater than or equal to calculated quench time</td>
<td>L</td>
<td>56, 113</td>
</tr>
<tr>
<td>66</td>
<td>Limit on rate of change of energy in poloidal field</td>
<td>L</td>
<td>65, 115</td>
</tr>
</tbody>
</table>

Table 2.4: Summary of constraint equations 51 onwards. Consistency equations are marked C, limit equations are marked L. Some (non-exhaustive) iteration variable numbers (see Tables 2.1 and 2.2) that directly affect the associated constraint equations are given, the one listed first being the most relevant.

### 2.6 Figures of Merit

In optimisation mode, PROCESS finds the self-consistent set of iteration variable values that maximises or minimises a certain function of them, known as the *figure of merit*. Several possible figures of merit are available, all of which are formulated in routine FUNFOM in source file evaluators.f90. Switch minmax is used to control which figure of merit is to be used, as summarised in Table 2.5. If the figure of merit is to be minimised, minmax should be positive, and if a maximised figure of merit is desired, minmax should be negative.

### 2.7 Scanning Variables

One of a number of variables can be scanned during the course of a PROCESS run. This option provides a method of determining the sensitivity of the results to different input assumptions. The user specifies which variable is to be scanned (see Table 2.6) and its required value at each point in the scan. Set input parameter isweep to the desired number of scan points. The scanned variable to use is defined by the value of nsweep, and the chosen variable’s values during the scan are set in array sweep.

Runs involving scans of this kind can only be performed in optimisation mode. The results from the previous scan point are used as the input to the next scan point. Routine SCAN in source file scan.f90 stores many of the output quantities in a separate output file called PLOT.DAT, which can be read by the utility program plot_sweep to produce graphical output (see Chapter 6).

It is also possible to scan a derived quantity that is not set directly by the user. For instance, one can scan the net electric power by constraining it to a specified value using the technique described in Section 2.5.2. The required net electric power, \( p_{\text{netelin}} \), is then used as the scanning variable.

For obvious reasons, the active scanning variable must not also be an active iteration variable.
<table>
<thead>
<tr>
<th>minmax</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>±1</td>
<td>plasma major radius</td>
</tr>
<tr>
<td>±2</td>
<td>not used</td>
</tr>
<tr>
<td>±3</td>
<td>neutron wall load</td>
</tr>
<tr>
<td>±4</td>
<td>total TF coil + PF coil power</td>
</tr>
<tr>
<td>±5</td>
<td>ratio of fusion power to injection power</td>
</tr>
<tr>
<td>±6</td>
<td>cost of electricity</td>
</tr>
</tbody>
</table>
| ±7     | \[
|       | \begin{cases}
|       | \text{direct cost} & \text{if } \text{ireactor} = 0 \\
|       | \text{constructed cost} & \text{otherwise}
|       | \end{cases}
| ±8     | aspect ratio                                         |
| ±9     | divertor heat load                                   |
| ±10    | toroidal field on axis                               |
| ±11    | injection power                                      |
| ±12    | hydrogen plant capital cost                          |
| ±13    | hydrogen production rate                             |
| ±14    | pulse length                                         |
| ±15    | plant availability factor (N.B. requires iavail>1)   |
| ±16    | minimise R0 and maximise burn time (linear combination) |
| ±17    | net electrical power                                 |

Table 2.5: Summary of the available figures of merit in PROCESS. If the figure of merit is to be minimised, \( \text{minmax} \) should be positive, and if a maximised figure of merit is desired, \( \text{minmax} \) should be negative.
<table>
<thead>
<tr>
<th>nsweep</th>
<th>scan variable</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>aspect</td>
<td>plasma aspect ratio</td>
</tr>
<tr>
<td>2</td>
<td>hldivlim</td>
<td>maximum divertor heat load</td>
</tr>
<tr>
<td>3</td>
<td>pnetelin</td>
<td>required net electric power</td>
</tr>
<tr>
<td>4</td>
<td>hfact</td>
<td>confinement time $H$-factor</td>
</tr>
<tr>
<td>5</td>
<td>oacdcp</td>
<td>overall current density in TF coil inboard leg</td>
</tr>
<tr>
<td>6</td>
<td>walalw</td>
<td>allowable wall load</td>
</tr>
<tr>
<td>7</td>
<td>beamfus0</td>
<td>beam-background fusion multiplier</td>
</tr>
<tr>
<td>8</td>
<td>fqval</td>
<td>$f$-value for fusion gain limit eqn</td>
</tr>
<tr>
<td>9</td>
<td>te</td>
<td>electron temperature</td>
</tr>
<tr>
<td>10</td>
<td>boundu(15)</td>
<td>upper bound on $f$-value $f_{vs}$</td>
</tr>
<tr>
<td>11</td>
<td>dbetan</td>
<td>beta $g$ coefficient</td>
</tr>
<tr>
<td>12</td>
<td>bscfmax</td>
<td>bootstrap current fraction (use negative values)</td>
</tr>
<tr>
<td>13</td>
<td>boundu(10)</td>
<td>upper bound on confinement time $H$-factor $h_{fact}$</td>
</tr>
<tr>
<td>14</td>
<td>fiooiic</td>
<td>$f$-value for TF coil operational current limit eqn</td>
</tr>
<tr>
<td>15</td>
<td>fjprot</td>
<td>$f$-value for TF coil current protection limit eqn</td>
</tr>
<tr>
<td>16</td>
<td>rmajor</td>
<td>plasma major radius</td>
</tr>
<tr>
<td>17</td>
<td>bmxlim</td>
<td>maximum toroidal field</td>
</tr>
<tr>
<td>18</td>
<td>gamma</td>
<td>maximum current drive gamma</td>
</tr>
<tr>
<td>19</td>
<td>boundl(16)</td>
<td>lower bound on central solenoid thickness $oh_{c\theta}$</td>
</tr>
<tr>
<td>20</td>
<td>tbrmm</td>
<td>minimum burn time (pulsed operation machine)</td>
</tr>
<tr>
<td>21</td>
<td>sigpfalw</td>
<td>allowable stress in the PF coils</td>
</tr>
<tr>
<td>22</td>
<td>cfactr</td>
<td>plant availability factor (N.B. requires $iavail=0$)</td>
</tr>
<tr>
<td>23</td>
<td>boundu(72)</td>
<td>upper bound on $f$-value $f_{ipir}$</td>
</tr>
<tr>
<td>24</td>
<td>powfmax</td>
<td>maximum fusion power</td>
</tr>
<tr>
<td>25</td>
<td>kappa</td>
<td>plasma elongation</td>
</tr>
<tr>
<td>26</td>
<td>triang</td>
<td>plasma triangularity</td>
</tr>
<tr>
<td>27</td>
<td>tbrmin</td>
<td>minimum tritium breeding ratio ($blktmodel=1$)</td>
</tr>
<tr>
<td>28</td>
<td>bt</td>
<td>toroidal field on axis</td>
</tr>
<tr>
<td>29</td>
<td>coreradius</td>
<td>normalised radius defining the core region</td>
</tr>
<tr>
<td>30</td>
<td>impvar</td>
<td>impurity fraction of element $imp_{var}$</td>
</tr>
<tr>
<td>31</td>
<td>taulimit</td>
<td>Taup/tau eff lower limit</td>
</tr>
</tbody>
</table>

Table 2.6: Summary of the scanning variables available in PROCESS.
Chapter 3

Physics, Engineering and Other Models

There are a great number of individual models within PROCESS, characterising many different aspects of a fusion power plant. Several of these will always be used by the code and so require no input by the user to activate them. However, in many cases there is a choice of model available, and each of these has its own user-controlled switches or flags. This chapter summarises some of these models, and indicates their location and interaction within the code, together with the relevant switch settings and required parameter values. The concepts used (iteration variables, constraint equations, etc.), and instructions on how to set switches, etc., are explained fully in Chapter 2.

It is essential for the user to refer to the variable descriptor file, vardes.html, this contains details of all the parameters within the code that can be changed by the user, in order to customise the machine modelled by PROCESS.

Disclaimer: Users are discouraged from using non-default or older models, as they are likely to be less well tested than the default models and should therefore be treated with caution.

3.1 Tokamak Power Plant

The default (and most detailed) power plant model in PROCESS is based on the tokamak magnetic-confinement fusion concept. This section describes the models relevant to such a plant in some detail, starting with the plasma at the centre of the fusion power core, and moving outwards to the external components, power subsystems and buildings. However, many of these models are also partly or wholly relevant for the other available machine types, especially outside the fusion power core. The specific details for these alternative models are introduced later in the Chapter.

3.1.1 Radial and vertical build

Figure 3.1 shows schematically the layout of a typical tokamak as modelled by PROCESS. This is the so-called ‘build’ of the machine — the relative locations of the major components. Their positions are referenced to the (R, Z) coordinate system, where R is the radial distance from the vertical centreline (axis) of the torus, and Z is the vertical distance from the equatorial midplane, about which the machine is assumed to be up-down symmetrical (by default; the vertical build is slightly different for single null plasma devices — see Figure 3.3). Components are often referred to as being ‘inboard’ or ‘outboard’, which simply means that they lie at a radius R less than or greater than $R_0$, respectively, where $R_0$ is the plasma major radius ($r_{major}$). For the sake of clarity the thicknesses are not drawn.
to scale, and the space labelled as the divertor does not indicate in any way the actual shape of that component.

The first wall, blanket, shield and vacuum vessel may be either D-shaped in cross-section, or each may be defined by two half-ellipses; compare their shapes in Figures 3.1 and 3.2. The choice between these two possibilities is set using input parameter \( fwbsshape \), which should be 1 for D-shaped, or 2 for ellipses.

![Diagram of the fusion power core of a typical tokamak power plant](image)

Figure 3.1: Schematic diagram of the fusion power core of a typical tokamak power plant modelled by PROCESS, showing the relative positions of the components. A double null plasma is assumed (\( snull=0 \)) — compare Figure 3.3, and the first wall, blanket, shield and vacuum vessel are D-shaped in cross-section (chosen by setting switch \( fwbsshape=1 \)) — compare Figure 3.2. Also shown are the code variables used to define the thicknesses of the components. The arrowed labels adjacent to the axes are the total ‘builds’ to that point. The precise locations and sizes of the PF coils are calculated within the code (see Section 3.1.7).

Most of the thicknesses shown in Figures 3.1 and 3.2 are input parameters, so are not changed during the course of the simulation. The rest are calculated by the code during execution. In addition, some of the component sizes can be used as iteration variables (see Section 2.4) to help in the optimisation.
Figure 3.2: Schematic diagram of the fusion power core of a typical tokamak power plant modelled by PROCESS. The first wall, blanket, shield and vacuum vessel cross-sectional shapes are each assumed to be defined by two ellipses (chosen by setting switch `fwbshape=2`) — compare Figure 3.1.
3.1.2 Plasma physics models

By default, the plasma is assumed to have an up-down asymmetric, single null configuration (although this can be changed if desired — see Section 3.1.5). A great number of physics models are coded within PROCESS to describe the behaviour of the plasma parameters such as its current, temperature, density, pressure, confinement etc., and also the various limits that define the stable operating domain.

3.1.2.1 Plasma geometry

The plasma (geometric) major radius $R_0$ ($r_{\text{major}}$) and aspect ratio $A$ ($\text{aspect}$) define the size of the plasma torus. The plasma minor radius $a$ ($r_{\text{minor}}$) is calculated from these values. The shape of the plasma cross-section is given by its last closed flux surface (LCFS) elongation $\kappa$ ($\text{kappa}$) and triangularity $\delta$ ($\text{triang}$), which can be scaled automatically with the aspect ratio if required using switch $\text{ishape}$:

- If $\text{ishape} = 0$, the input values for $\text{kappa}$ and $\text{triang}$ are used directly.
- If $\text{ishape} = 1$, the following scaling is used, which is suitable for low aspect ratio machines ($\epsilon = 1/A$) [2]:

$$
\kappa = 2.05 \left(1 + 0.44 \epsilon^{2.1}\right) 
$$

$$
\delta = 0.53 \left(1 + 0.77 \epsilon^{3}\right) 
$$

- If $\text{ishape} = 2$, the Zohm ITER scaling [9] is used to calculate the elongation:

$$
\kappa = F_{kz} \times \text{minimum} \left(2.0, 1.5 + \frac{0.5}{A - 1}\right) 
$$

where input variable $f_{kzohm} = F_{kz}$ may be used to adjust the scaling, while the input value of the triangularity is used unchanged.

If $\text{ishape} = 0$, 1 or 2, the values for the plasma shaping parameters at the 95% flux surface are calculated as follows [10]:

$$
\kappa_{95} = \frac{\kappa}{1.12} 
$$

$$
\delta_{95} = \frac{\delta}{1.5} 
$$

- If $\text{ishape} = 3$, the Zohm ITER scaling [9] is used to calculate the elongation (as for $\text{ishape} = 2$ above), but the triangularity at the 95% flux surface is input via variable $\text{triang}_{95}$, and the LCFS triangularity $\text{triang}$ is calculated from it, rather than the other way round.

- Finally, if $\text{ishape} = 4$, the 95% flux surface values $\kappa_{95}$ and $\text{triang}_{95}$ are both used as inputs, and the LCFS values are calculated from them by inverting Equations 3.4 and 3.5.

A constraint relating to the plasma’s vertical stability may be turned on if required. In principle, the inner surface of the outboard shield could be used as the location of a conducting shell which should mitigate the vertical displacement growth rate of plasmas with significant elongation [11]. The maximum distance $r_{\text{shell, max}}$ of this shell from the centre of the plasma may be set using input
Figure 3.3: Schematic diagram of the fusion power core of a typical tokamak power plant modelled by PROCESS, showing the relative positions of the components. A single null plasma is assumed ($s_{null}=1$) — compare Figure 3.2. The radial build is the same as for a double null configuration; shown along the vertical axis are the code variables used to define the vertical thicknesses of the components. The arrowed labels adjacent to the axis are the total ‘builds’ (distance from the midplane, $Z=0$) to that point. The precise locations and sizes of the PF coils are calculated within the code (see Section 3.1.7).
parameter $c_{\text{wrmax}}$, such that $r_{\text{shell, max}} = c_{\text{wrmax}}*r_{\text{minor}}$. Constraint equation no. 23 should be turned on with iteration variable no. 104 ($f_{\text{cwr}}$) to enforce this. (A scaling of $c_{\text{wrmax}}$ with elongation should be available shortly.)

The plasma surface area, cross-sectional area and volume are calculated using formulations that approximate the LCFS as a revolution of two arcs which intersect the plasma X-points and the plasma midplane outer and inner radii. (This is a reasonable assumption for double-null diverted plasmas, but will be inaccurate for single-null plasmas, $s_{\text{null}} = 1$.) Switch $\text{igeom}$ determines whether an old method is used ($\text{igeom} = 0$) or calculations based on a more recent derivation ($\text{igeom} = 1$) [12].

### 3.1.2.2 Fusion reactions

The most likely fusion reaction to be utilised in a power plant is the deuterium-tritium reaction:

$$D + T \implies ^4\text{He} + n + 17.6 \text{ MeV} \quad (3.6)$$

20% of the energy produced is given to the alpha particles ($^4\text{He}$), a fraction of which remain (c.f. $f_{\text{alpha}}$, Section sec:corepower) within the plasma and thermalise (slow down) due to collisions, thus heating the plasma. The remaining 80% is carried away by the neutrons, which deposit their energy within the blanket and shield.

PROCESS can also model D-$^3\text{He}$ power plants, which utilise the following primary fusion reaction:

$$D + ^3\text{He} \implies ^4\text{He} + p + 18.3 \text{ MeV} \quad (3.7)$$

The fusion reaction rate is significantly different to that for D-T fusion, and the power flow from the plasma is modified since charged particles are produced rather than neutrons. Because only charged particles (which remain in the plasma) are produced by this reaction, the whole of the fusion power is used to heat the plasma. Useful energy is extracted from the plasma since the radiation power produced is very high, and this can be converted to electricity in a number of ways.

Since the temperature required to ignite the D-$^3\text{He}$ reaction is considerably higher than that for D-T, it is necessary to take into account the following D-D reactions, which have significant reaction rates at such temperatures:

$$D + D \implies ^3\text{He} + n + 3.27 \text{ MeV} \quad (3.8)$$

$$D + D \implies T + p + 4.03 \text{ MeV} \quad (3.9)$$

Also, as tritium is produced by the latter reaction, D-T fusion is also possible. As a result, there is still a small amount of neutron power extracted from the plasma.

Pure D-$^3\text{He}$ tokamak power plants do not include blankets, because of the near absence of neutrons leaving the plasma, and the fact that no tritium needs to be produced for fuel.

The contributions from all four of the above fusion reactions are included in the total fusion power production calculation. The fusion reaction rates are calculated using the parametrizations in [13], integrated over the plasma profiles (correctly, with or without pedestals).

The fractional composition of the “fuel” ions (D, T and $^3\text{He}$) is controlled using the three variables $f_{\text{deut}}$, $f_{\text{trit}}$ and $f_{\text{he3}}$, respectively:

\[
\begin{align*}
    n_{\text{fuel}} &= n_D + n_T + n_{^3\text{He}} \quad \text{particles/m}^3 \\
    n_D &= f_{\text{deut}}n_{\text{fuel}} \\
    n_T &= f_{\text{trit}}n_{\text{fuel}} \\
    n_{^3\text{He}} &= f_{\text{he3}}n_{\text{fuel}}
\end{align*}
\]

PROCESS checks that $f_{\text{deut}} + f_{\text{trit}} + f_{\text{he3}} = 1.0$, and stops with an error message otherwise.
3.1.2.3 Plasma profiles

If switch $\text{ipedestal} = 0$, the plasma profiles are assumed to be parabolic, i.e. they are of the form

\begin{align}
\text{Density: } n(\rho) &= n_0 \left(1 - \rho^2\right)^{\alpha_n} \\
\text{Temperature: } T(\rho) &= T_0 \left(1 - \rho^2\right)^{\alpha_T} \\
\text{Current: } J(r) &= J_0 \left(1 - \rho^2\right)^{\alpha_J}
\end{align}

(3.10) (3.11) (3.12)

where $\rho = r/a$, and $a$ is the plasma minor radius. This gives volume-averaged values $\langle n \rangle = n_0/(1+\alpha_n)$, and line-averaged values $\bar{n} \sim n_0/\sqrt{(1+\alpha_n)}$, etc. These volume- and line-averages are used throughout the code along with the profile indices $\alpha$, in the various physics models, many of which are fits to theory-based or empirical scalings. Thus, the plasma model in PROCESS may be described as “1-D”.

The relevant profile index variables are $\text{alphan}$, $\text{alphat}$ and $\text{alphaj}$, respectively (see Section 3.1.2.9).

However, by default, $\text{ipedestal} = 1$ which allows the density and temperature profiles to include a pedestal, using the forms specified in [14]:

\begin{align}
\text{density: } n(\rho) &= \begin{cases} 
 n_{\text{ped}} + (n_0 - n_{\text{ped}}) \left(1 - \frac{\rho^2}{\rho_{\text{ped},n}^2}\right)^{\alpha_n} & 0 \leq \rho \leq \rho_{\text{ped},n} \\
 n_{\text{sep}} + (n_{\text{ped}} - n_{\text{sep}}) \left(1 - \frac{\rho}{1 - \rho_{\text{ped},n}}\right) & \rho_{\text{ped},n} < \rho \leq 1
\end{cases} \\
\text{temperature: } T(\rho) &= \begin{cases} 
 T_{\text{ped}} + (T_0 - T_{\text{ped}}) \left(1 - \frac{\rho_{\beta_T}}{\rho_{\text{ped},T}^{\beta_T}}\right)^{\alpha_T} & 0 \leq \rho \leq \rho_{\text{ped},T} \\
 T_{\text{sep}} + (T_{\text{ped}} - T_{\text{sep}}) \left(1 - \frac{\rho}{1 - \rho_{\text{ped},T}}\right) & \rho_{\text{ped},T} < \rho \leq 1
\end{cases}
\end{align}

(3.13) (3.14)

Subscripts 0, $\text{ped}$ and $\text{sep}$, denote values at the centre ($\rho = 0$), the pedestal ($\rho = \rho_{\text{ped}}$) and the separatrix ($\rho = 1$), respectively. The density and temperature peaking parameters $\alpha_n$ and $\alpha_T$ as well as the second exponent $\beta_T$ (input parameter $\text{tbeta}$, not to be confused with the plasma beta) in the temperature profile can be chosen by the user, as can the pedestal heights and the values at the separatrix ($n_{\text{ped}}$, $n_{\text{sep}}$ for the electron density, and $T_{\text{ped}}$, $T_{\text{sep}}$ for the electron temperature; the ion equivalents are scaled from the electron values by the ratio of the volume-averaged values).

The density at the centre is given by

$$n_0 = \frac{1}{3\rho_{\text{ped},n}^2} \left[3\langle n \rangle (1 + \alpha_n) + n_{\text{sep}} (1 + \alpha_n) (-2 + \rho_{\text{ped},n} + \rho_{\text{ped},n}^2)ight.$$  
$$- n_{\text{ped}} (1 + \alpha_n) (1 + \rho_{\text{ped},n}) + (\alpha_n - 2) \rho_{\text{ped},n}^2 \left.\right]\]$$

(3.15)

where $\langle n \rangle$ is the volume-averaged density. The temperature at the centre is given by

$$T_0 = T_{\text{ped}} + \gamma \left[ T_{\text{ped}} \rho_{\text{ped},T}^2 - \langle T \rangle + \frac{1}{3} (1 - \rho_{\text{ped},T}) \left(1 + 2\rho_{\text{ped},T} \right) T_{\text{ped}} + (2 + \rho_{\text{ped},T}) T_{\text{sep}} \right]$$

(3.16)

with

$$\gamma = \begin{cases} 
 -\Gamma(1 + \alpha_T + 2/\beta_T) & \text{for integer } \alpha_T \\
 \frac{\rho_{\text{ped},T}^2}{\Gamma(1 + \alpha_T)} \Gamma((2 + \beta_T)/\beta_T) & \\
 \frac{\Gamma(-\alpha_T) \sin(\pi\alpha) \Gamma((1 + \alpha_T + 2/\beta_T)}}{\pi \rho_{\text{ped},T}^{\beta_T} \Gamma((2 + \beta_T)/\beta_T)} & \text{for non-integer } \alpha_T
\end{cases}$$

(3.17)

where $\Gamma$ is the gamma function.
Note that density and temperature can have different pedestal positions \( \rho_{\text{ped},n}(r_{\text{hopedn}}) \) and \( \rho_{\text{ped},T}(r_{\text{hopedt}}) \) in agreement with simulations.

The pedestal density can be set directly (if \text{iscdens}=1), or as a fraction of the Greenwald density (if \text{iscdens}=1). The default fraction is 0.8 [15].

### 3.1.2.4 Beta limits

The plasma beta limit [16][17] is given by

\[
\langle \beta \rangle < g \frac{I(\text{MA})}{a(\text{m}) B_0(T)}
\]

where \( B_0 \) is the axial vacuum toroidal field, and \( \beta \) is defined with respect to the total equilibrium \( B \)-field [17]. The beta coefficient \( g \) is set using input parameter \text{dnbeta} (but see Section 3.1.2.9). To apply the beta limit, constraint equation no. 24 should be turned on with iteration variable no. 36 (\text{fbetatry}). The limit can be applied to either the total plasma beta, in which case switch \text{iculbl} should be set to 0, to only the thermal component of the plasma beta, in which case \text{iculbl} should be set to 1, or to the thermal plus neutral beam components, in which case \text{iculbl} should be set to 2.

#### Aspect ratio scaling of beta \( g \) coefficient

Switch \text{gtscale} determines whether the beta \( g \) coefficient \text{dnbeta} should scale with aspect ratio (\text{gtscale} \neq 0), or be fixed at the input value (\text{gtscale} = 0). Note that \text{gtscale} is over-ridden if \text{iprofile} = 1 (see Section 3.1.2.9).

#### Limiting \( \epsilon, \beta_p \)

To apply a limit to the value of \( \epsilon, \beta_p \), where \( \epsilon = a/R \) is the inverse aspect ratio, constraint equation no. 6 should be turned on with iteration variable no. 8 (\text{fbeta}). The limiting value of \( \epsilon, \beta_p \) should be set using input parameter \text{epbetmax}.

### 3.1.2.5 Fast alpha pressure contribution

Switch \text{ifalphap} may be used to select the model used to calculate the pressure contribution from the fast alpha particles:

\[
\frac{\beta_{\alpha}}{\beta_{th}} = 0.29 \left( \langle T_{i0} \rangle - 0.37 \right) \left( \frac{n_{DT}}{n_e} \right)^2 \quad \text{ifalphap} = 0 \quad [16] \tag{3.19}
\]

\[
\frac{\beta_{\alpha}}{\beta_{th}} = 0.26 \big( \langle T_{i0} \rangle - 0.65 \big)^{0.5} \left( \frac{n_{DT}}{n_e} \right)^2 \quad \text{ifalphap} = 1 \quad \text{(default)} \quad [18] \tag{3.20}
\]

The latter model is a better estimate at higher temperatures.

### 3.1.2.6 Density limits

Several density limit models [17] are available in \text{PROCESS}. These are calculated in routine \text{CULDLM}, which is called by \text{PHYSICS}. To enforce any of these limits, turn on constraint equation no. 5 with iteration variable no. 9 (\text{fdene}). In addition, switch \text{idensl} must be set to the relevant value, as follows:-
3.1.2.7 Impurities and radiation

The impurity radiation model in PROCESS can be chosen using the switch `imprad_model`, where `imprad_model = 0` gives the original ITER 1989 model [19], and `imprad_model = 1` uses a multi-impurity model which integrates the radiation contributions over an arbitrary choice of density and temperature profiles [22].

If `imprad_model = 0`, the impurity species and fractions are chosen using input parameters `impc`, `impo`, `fbfe`, `cfe0` and `zfear`; see the variable descriptor file for more details.

If `imprad_model = 1`, the impurity number density fractions relative to the electron density are set using input array `fimp(1 to nimp)`, where `nimp = 14` is the number of impurity species in the radiation model. The available impurities are as follows:

1. Hydrogen (fraction calculated by code)
2. Helium (fraction calculated by code)
3. Beryllium
4. Carbon
5. Nitrogen
6. Oxygen
7. Neon
8. Silicon
9. Argon
10. Iron
11. Nickel
12. Krypton
13. Xenon
14. Tungsten
As stated above, the number density fractions for hydrogen (all isotopes) and helium need not be set, as they are calculated by the code to ensure that plasma quasi-neutrality is maintained, and taking into account the fuel ratios $f_{\text{deut}}$, $f_{\text{trit}}$ and $f_{\text{he3}}$ (see Section 3.1.2.2), and the alpha particle fraction $r_{\text{alpne}}$ which may be input by the user.

The impurity fraction of one of the elements listed in array $f_{\text{imp}}$ may be used as an iteration variable (although it will only have any effect if $\text{imprad\_model} = 1$). The element to use is specified using input parameter $\text{impvar}$, which may be set to a value between 3 and $\text{nimp}$, and the initial estimate to use for the element’s impurity fraction must be set using iteration variable no. 102 ($f_{\text{impvar}}$).

The synchrotron radiation power $[23, 24]$ is assumed to originate from the plasma core. The wall reflection factor $s_{\text{sync}}$ may be set by the user.

By changing the input parameter $\text{coreradius}$, the user may set the normalised radius defining the core region ($\text{imprad\_model} = 1$ only). Only the impurity and synchrotron radiation from the core region affects the confinement scaling (but see the $\text{iradloss}$ description in Section 3.1.2.10). Figure 3.4 elucidates the radiation power contributions, while Figures 3.5 and ?? show how the radiation power propagates through the fusion power core.

![Schematic diagram of the radiation power contributions and how they are split between core and edge radiation, for $\text{imprad\_model} = 1$.](image)

Constraint equation no. 17 with iteration variable no. 28 ($\text{fradpwr}$) ensures that the calculated total radiation power does not exceed the total power available that can be converted to radiation (i.e. the sum of the fusion alpha power, other charged particle fusion power, auxiliary injected power and the ohmic power). This constraint should always be turned on.

### 3.1.2.8 Plasma current scaling laws

A number of plasma current scaling laws exploiting the inverse relationship between plasma current and edge safety factor $q_0$ $[17]$ are available in PROCESS. These are calculated in routine CULCUR, which is called by PHYSICS. Flag $\text{icurr}$ must be set to the relevant value, as follows:-

- $\text{icurr} = 1$: Peng analytic fit
- $\text{icurr} = 2$: Peng double null divertor scaling (ST) $[2]$
- $\text{icurr} = 3$: Simple ITER scaling
- $\text{icurr} = 4$: Revised ITER scaling $[25]$
- $\text{icurr} = 5$: Todd empirical scaling, I
- $\text{icurr} = 6$: Todd empirical scaling, II
icurr = 7: Connor-Hastie model

3.1.2.9 Plasma current profile consistency

Self-consistency between the plasma current profile parameters [10] can be enforced by setting switch iprofile to 1. This ensures that the current profile peaking factor $\alpha_{j}$ is consistent with the input values for the safety factor on axis and at the plasma edge ($q_0$ and $q_e$, respectively), the plasma internal inductance $l_i$ is consistent with this $\alpha_j$, and the beta $g$ coefficient $\frac{d\beta}{ds}$ scales with $l_i$.

It is recommended that current scaling law icurr = 4 is used if iprofile = 1. Switch gtscale is over-ridden if iprofile = 1.

3.1.2.10 Confinement time scaling laws

The particle transport loss power in Watts/m$^3$ from the plasma is defined as

$$P_{\text{loss}} = \frac{W}{\tau_E} \tag{3.21}$$

where the plasma stored energy per unit volume $W$ is given by

$$W = \frac{3}{2} n_e T$$

with particle number density $n$ in m$^{-3}$ and particle temperature $T$ in eV. PROCESS assumes that the electron and ion energy confinement times $\tau_E$ are equal.

Many energy confinement time scaling laws are present within PROCESS, for tokamaks, RFPs or stellarators. These are calculated in routine PCOND. The value of isc determines which of the scalings is used in the plasma energy balance calculation (Section 3.1.2.11). Table 3.1 summarises the available scaling laws. The most commonly used is the so-called IPB98(y;2) scaling.

Effect of radiation on energy confinement

Published confinement scalings are all based on low radiation pulses. A power plant will certainly be a high radiation machine — both in the core, due to bremsstrahlung and synchrotron radiation, and in the edge due to impurity seeding. The scaling data do not predict this radiation — that needs to be done by the radiation model. However, if the transport is very “stiff”, as predicted by some models, then the additional radiation causes an almost equal drop in power transported by ions and electrons, leaving the confinement nearly unchanged.

To allow for these uncertainties, three options are available, using the switch iradloss. In each case, the particle transport loss power $P_{\text{loss}}$ (Equation 3.21), referred to in the code as pscaling, is derived directly from the energy confinement scaling law (see 3.1.2.10 and Figure 3.4 above).

iradloss = 0: Total power lost is scaling power plus radiation:

$$\text{pscaling} + \text{pradpv} = \text{falpha*palppv} + \text{pchargepv} + \text{pohmpv} + \text{pinjmw/vol}$$

iradloss = 1: Total power lost is scaling power plus core radiation only:

$$\text{pscaling} + \text{pcoreradpv} = \text{falpha*palppv} + \text{pchargepv} + \text{pohmpv} + \text{pinjmw/vol}$$
<table>
<thead>
<tr>
<th>isc</th>
<th>scaling law</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Neo-Alcator (ohmic)</td>
<td>[16]</td>
</tr>
<tr>
<td>2</td>
<td>Mirnov (H-mode)</td>
<td>[16]</td>
</tr>
<tr>
<td>3</td>
<td>Merezhkin-Muhkovatov (L-mode)</td>
<td>[16]</td>
</tr>
<tr>
<td>4</td>
<td>Shimomura (H-mode)</td>
<td>JAERI-M 87-080 (1987)</td>
</tr>
<tr>
<td>5</td>
<td>Kaye-Goldston (L-mode)</td>
<td>Nuclear Fusion 25 (1985) p.65</td>
</tr>
<tr>
<td>7</td>
<td>ITER 89-O (L-mode)</td>
<td>[16]</td>
</tr>
<tr>
<td>10</td>
<td>T10 (L-mode)</td>
<td>[16]</td>
</tr>
<tr>
<td>11</td>
<td>JAERI-88 (L-mode)</td>
<td>JAERI-M 88-068 (1988)</td>
</tr>
<tr>
<td>13</td>
<td>ITER H90-P (H-mode)</td>
<td></td>
</tr>
<tr>
<td>14</td>
<td>ITER Mix (minimum of 6 and 7)</td>
<td></td>
</tr>
<tr>
<td>15</td>
<td>Riedel (L-mode)</td>
<td></td>
</tr>
<tr>
<td>16</td>
<td>Christiansen et al. (L-mode)</td>
<td>JET Report JET-P (1991) 03</td>
</tr>
<tr>
<td>17</td>
<td>Lackner-Gottardi (L-mode)</td>
<td>Nuclear Fusion 30 (1990) p.767</td>
</tr>
<tr>
<td>18</td>
<td>Neo-Kaye (L-mode)</td>
<td>[16]</td>
</tr>
<tr>
<td>19</td>
<td>Riedel (H-mode)</td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>Large Helical Device (stellarator)</td>
<td>Nuclear Fusion 30 (1990) p.11</td>
</tr>
<tr>
<td>23</td>
<td>Lackner-Gottardi (stellarator)</td>
<td>Nuclear Fusion 30 (1990) p.767</td>
</tr>
<tr>
<td>25</td>
<td>TITAN (RFP)</td>
<td>TITAN RFP Fusion Reactor Study, Scoping Phase Report UCLA-PPG-1100, page 5–9, Jan 1987</td>
</tr>
<tr>
<td>26</td>
<td>ITER H-97P ELM-free (H-mode)</td>
<td>J. G. Cordey et al., EPS Berchtesgaden, 1997</td>
</tr>
<tr>
<td>27</td>
<td>ITER H-97P ELMy (H-mode)</td>
<td>J. G. Cordey et al., EPS Berchtesgaden, 1997</td>
</tr>
<tr>
<td>29</td>
<td>Valovic modified ELMy (H-mode)</td>
<td></td>
</tr>
<tr>
<td>30</td>
<td>Kaye PPPL April 98 (L-mode)</td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>ITERH-PB98P(y) (H-mode)</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>IPB98(y,2) (H-mode)</td>
<td>Nuclear Fusion 39 (1999) p.2175</td>
</tr>
<tr>
<td>36</td>
<td>IPB98(y,4) (H-mode)</td>
<td>Nuclear Fusion 39 (1999) p.2175</td>
</tr>
<tr>
<td>37</td>
<td>ISS95 (stellarator)</td>
<td>Nuclear Fusion 36 (1996) p.1063</td>
</tr>
<tr>
<td>38</td>
<td>ISS04 (stellarator)</td>
<td>Nuclear Fusion 45 (2005) p.1684</td>
</tr>
</tbody>
</table>

Table 3.1: Summary of the energy confinement time scaling laws in PROCESS.
iradloss = 2: Total power lost is scaling power only, with no additional allowance for radiation. This is not recommended for power plant models.

\[ p\text{scaling} = f\alpha p\text{alppv} + p\text{chargepv} + p\text{ohmpv} + p\text{injmw/vol} \]

### 3.1.2.11 Plasma core power balance

Figure 3.5 shows the flow of power as calculated by the code. (Figure 3.9 shows an alternative view of the power transfer outside the reactor core.)

![Power flows diagram]

The primary sources of power are the fusion reactions themselves, ohmic power due to resistive heating within the plasma, and any auxiliary power provided for heating and current drive (see Section 3.1.9). The power carried by the fusion-generated neutrons is lost from the plasma, but is deposited in the surrounding material (see Section 3.1.12). A fraction \( f\alpha \) of the alpha particle power is assumed to stay within the plasma core to contribute to the plasma power balance. The sum of this core alpha power, any power carried by non-alpha charged particles, the ohmic power and any injected power, is converted into charged particle transport power (\( P_{\text{loss}} \) in Equation 3.21) plus core radiation power (see Section 3.1.2.7), as shown in Figure 3.5.

The core power balance calculation is turned on using constraint equation no. 2 (which should therefore always be used).

### 3.1.2.12 Bootstrap current scalings

The fraction of the plasma current provided by the so-called bootstrap effect can be either input into the code directly, or calculated using one of four methods, as summarised here. Note that methods ibss = 1 to 3 use fits to parabolic density and temperature profiles, and do not take into account the existence of pedestals (ipedestal = 1), whereas the Sauter et al. scaling (ibss = 4) allows general profiles to be used.

- Direct input: To input the bootstrap current fraction directly, set bscfmax to \((-1)\) times the required value.
• ITER scaling [16]: To use the ITER scaling method for the bootstrap current fraction, set $ibss = 1$, and $bscfmax$ to the maximum required bootstrap current fraction ($\leq 1$). This method is valid at high aspect ratio only.

• General scaling [26]: To use a more general scaling method, set $ibss = 2$, and $bscfmax$ to the maximum required bootstrap current fraction ($\leq 1$).

• Numerically fitted scaling [27]: To use a numerically fitted scaling method, valid for all aspect ratios, set $ibss = 3$, and $bscfmax$ to the maximum required bootstrap current fraction ($\leq 1$).

• Sauter, Angioni and Lin-Liu scaling [28, 29]: Set $ibss = 4$, and $bscfmax$ to the maximum required bootstrap current fraction ($\leq 1$).

### 3.1.2.13 L-H power threshold scalings

Transitions from a standard confinement mode (L-mode) to an improved confinement regime (H-mode), called L-H transitions, are observed in most tokamaks. A range of scaling laws are available that provide estimates of the auxiliary power required to initiate these transitions, via extrapolations from present-day devices. PROCESS calculates these power threshold values for the scaling laws listed in Table 3.2 in routine PTHRESH.

To enforce one of these scalings, use input parameter $ilhthresh$ to select the scaling to use, and turn on constraint equation no. 23 with iteration variable no. 104 ($flhthresh$). By default, this will ensure that the power reaching the divertor is at least equal to the threshold power calculated for the chosen scaling, which is a necessary condition for H-mode. However, for L-mode it might be appropriate to set $boundl(104) = 0.001$, $boundu(104) = 1.0$, which will ensure that the power threshold does not exceed the calculated value, and therefore the machine cannot reach H-mode.

<table>
<thead>
<tr>
<th>name</th>
<th>reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>(2) 1996 ITER scaling: upper bound</td>
<td></td>
</tr>
<tr>
<td>(3) 1996 ITER scaling: lower bound</td>
<td></td>
</tr>
<tr>
<td>(5) 1997 ITER scaling, including elongation</td>
<td></td>
</tr>
<tr>
<td>(7) 2008 Martin scaling: 95% upper bound</td>
<td></td>
</tr>
<tr>
<td>(8) 2008 Martin scaling: 95% lower bound</td>
<td></td>
</tr>
</tbody>
</table>

**Table 3.2: Summary of the L-H power threshold scalings implemented in PROCESS.**

### 3.1.2.14 Other plasma physics options

**Neo-classical correction effects**

Switch $ires$ controls whether neo-classical (trapped particle) effects [30] are included in the calculation of the plasma resistance and ohmic heating power in routine POHM, which is called by routine PHYSICS. If $ires = 1$, these effects are included. Note that the scaling used is only valid for aspect ratios between 2.5 and 4, and it is possible for the plasma resistance to be wrongly calculated as negative if $ires = 1$ and the aspect ratio is too high.
Inverse quadrature in $\tau_E$ scaling laws

Switch $i_{\text{invqd}}$ determines whether the energy confinement time scaling laws (see Section 3.1.2.10) due to Kaye-Goldston ($isc = 5$) and Goldston ($isc = 9$) should include an inverse quadrature scaling with the Neo-Alcator result ($isc = 1$). A value $i_{\text{invqd}} = 1$ includes this scaling.

Plasma / first wall gap

The region directly outside the last closed flux surface of the core plasma is known as the scrape-off layer, and contains no structural material. Plasma entering this region is not confined and is removed by the divertor. PROCESS treats the scrape-off layer merely as a gap. Switch $isc_{\text{rp}}$ determines whether the inboard and outboard gaps should be calculated as 10% of the plasma minor radius ($isc_{\text{rp}} = 0$), or set equal to the input values $\text{scrapli}$ and $\text{scraplo}$ ($isc_{\text{rp}} = 1$).

3.1.3 Armour, first wall and breeding blanket

The surface facing the plasma is a thin layer of a material highly resistant to melting and erosion, such as tungsten, referred to "armour". It is cooled by conduction to the first wall underneath.

The first wall sits behind the armour, and is dedicated to removing the heat landing on the armour. It does not breed tritium. Due to the hostile environment the first wall and armour have only a short lifetime and therefore need to be replaced regularly. It is cooled either by gaseous helium or by pressurised liquid water, depending on the selection of blanket type using the switch $\text{blkttype}$ – see Section 3.1.3.1.

Wall load calculation

Switch $i_{\text{walld}}$ determines whether the neutron wall load (power per unit area) should be calculated using the plasma surface area ($i_{\text{walld}} = 1$) or the first wall area ($i_{\text{walld}} = 2$) as the denominator. In the former case, input parameter $ff_{\text{wal}}$ (default value 0.92) can be used to scale the neutron power reaching the first wall.

The breeding blanket performs a number of tasks. An incoming neutron from a deuterium-tritium (D-T) fusion reaction in the plasma loses energy in the blanket. This energy is removed by the blanket coolant and used to produce electricity. The neutron may also react with a lithium nucleus present in the blanket to produce ("breed") a tritium nucleus which can be re-used as fuel. The competing requirements of heating and tritium synthesis mean that a neutron multiplier must be present, to ensure balance between tritium destruction and creation. The blanket therefore contains beryllium to fulfil this purpose. As with the first wall, the blanket has a relatively short lifetime because of the high neutron fluence.

3.1.3.1 Blanket model options

The models used for the thermodynamics of the first wall, the profile of deposition of the neutron energy, tritium breeding, and conversion of heat to electricity have been revised extensively.

$i_{\text{blanket}}$: This switch selects between different types of blanket.

In the CCFE HCPB (helium-cooled pebble bed) model ($i_{\text{blanket}} = 1$), the energy deposition in the armour and first wall, blanket and shield are calculated using parametric fits to an MCNP
neutron and photon transport model of a sector of a tokamak. The blanket contains lithium orthosilicate Li$_4$SiO$_4$, titanium beryllide TiBe$_{12}$, helium and Eurofer steel.

The KIT HCPB model (i.blanket = 2) allows the energy multiplication factor $emult$, the shielding requirements and tritium breeding ratio to be calculated self-consistently with the blanket and shielding materials and sub-assembly thicknesses, and for constraints to be applied to satisfy the engineering requirements. For further details of this model, see Section 3.1.3.2.

The CCFE HCPB model with tritium breeding ratio (i.blanket = 3) has the features of the CCFE HCPB model above, with a set of fitting functions for calculating tritium breeding ratio (TBR). It requires a choice of i.blkth, specifying a THIN, MEDIUM or THICK blanket. This fixes the values of inboard and outboard blanket thickness, and the initial values of first wall thickness (3 cm) and first wall armour (3 mm). Note that these last two can be modified by the first wall thermohydraulic module, in which case the output will not be fully self-consistent. The lithium-6 enrichment and the breeder fraction (Li$_4$SiO$_4$/(Be$_{12}$Ti+Li$_4$SiO$_4$) by volume) are available as iteration variables, and the minimum TBR can be set as a constraint. The maximum values of TBR achievable are as follows:


`secondary_cycle`: This switch controls how the coolant pumping power in the first wall and blanket is determined, and also how the calculation of the plant’s thermal to electric conversion efficiency (the secondary cycle thermal efficiency) proceeds. See Section 3.1.12.

### 3.1.3.2 KIT Blanket neutronics model

The model used if blktmodel = 1 is based on the Helium-Cooled Pebble Bed (HCPB) blanket concept developed by KIT (a second advanced model — Helium-Cooled Lithium Lead, HCLL — will be implemented in due course). The blanket, shield and vacuum vessel are segmented radially into a number of sub-assemblies. Moving in the direction away from the plasma/first wall, these are:

- **Breeding Zone (BZ)** (which includes the first wall), with radial thicknesses (inboard and outboard, respectively) f.with+blbuit.th, f.woth+blbuito.th. This consists of beryllium (with fraction by volume f.blbe), breeder material (f.blbreed), steel (f.blss) and helium coolant. Three forms of breeder material are available: lithium orthosilicate (Li$_4$SiO$_4$) (chosen by setting breedmat = 1), lithium metatitanate (Li$_2$TiO$_3$) (breedmat = 2) or lithium zirconate (Li$_2$ZrO$_3$) (breedmat = 3). The $^6$Li enrichment percentage may be modified from the default 30% using input parameter li6enrich.

- **Box Manifold (BM)**, with radial thicknesses (inboard and outboard, respectively) blbmi.th, blbmoth and helium fractions f.blhebmi, f.blhebmo (the rest being steel).

- **Back Plate (BP)**, with radial thicknesses (inboard and outboard, respectively) blbpith, blbpoth and helium fractions f.blhebpi, f.blhebpo (the rest being steel).

Together, the BZ, BM and BP make up the ‘blanket’, with total radial thicknesses blnkith (inboard) and blnkoth (outboard), and void (coolant) fraction vf.blkt. Note that these quantities are calculated from the sub-assembly values if blktmodel > 0, rather than being input parameters.

- **Low Temperature Shield and Vacuum Vessel** (lumped together for these calculations), with radial thicknesses (inboard and outboard, respectively) shldith+ddwi, shldoth+ddwi and water coolant fraction vf.shld (the rest being assumed to be steel for its mass calculation; the neutronics
model assumes that the shield contains 2% boron as a neutron absorber, but this material is not
explicitly mentioned elsewhere in the code — so its cost is not calculated, for example).

N.B. The fact that water is assumed to be the coolant in the shield, whereas helium is the coolant
in the blanket, leads to an inconsistency when specifying the coolant type via switch coolwh (see
Section 2.1.3.1). At present we mitigate this by forcing coolwh=2 (making water the coolant), as
in this case the coolant mass and pumping costs are higher, giving the more pessimistic solution
with regards to costs.

A few other input parameters are useful for tuning purposes, as follows:

- **fvolsi** and **fvolso** are the area (and volume) coverage factors for the inboard and outboard
  shields, respectively.
- fvolw is a multiplier for the volume of the vacuum vessel, used in the item’s mass calculation
to account for ports, etc.
- npdiv is the number of divertor ports, used in the calculation of the tritium breeding ratio.
- nphcdin and nphcdout are the number of heating/current drive ports on the inboard and
  outboard sides, respectively, used in the calculation of the tritium breeding ratio. These may be
  either ‘small’ (hcdportsi = 1) or ‘large’ (hcdportso = 2).
- wallpf is the neutron wall load peaking factor (maximum/mean), used in the calculation of the
  blanket lifetime.
- ucblbreed is the unit cost ($/kg) of the breeder material.

KIT model outputs and available constraints

The KIT blanket neutronics model provides the following outputs:

- The total nuclear power deposited in the blanket and shield, pnucblkt and pnucshld, respectively,
  and the energy multiplication factor in the blanket, emult are calculated.
- The tritium breeding ratio, tbr. This can be constrained to be no less than a certain value tbrmin
  for tritium self-sufficiency by turning on constraint equation no. 52 with iteration variable no.
  89 (ftbr). The inboard and outboard blanket BZ thicknesses, blbuith and blbuoth can also
  be used as iteration variables (90 and 91, respectively) to help the constraint to be met.
- The tritium production rate in grammes/day is calculated.
- The fast neutron fluence (neutrons/m²) on the TF coils is calculated. The peak value of this
  quantity may be constrained to be no more than a maximum value nflutfmax by turning on
  constraint equation no. 53 with iteration variable no. 92 (fflutf). The inboard and outboard
  shield thicknesses, shldith and shldoth can also be used as iteration variables (93 and 94,
  respectively) to help the constraint to be met. (Note that in this calculation the TF coil case
  surrounding the superconductor winding pack is ignored.)
- The nuclear heating power (MW/m³) on the inboard and outboard TF coils is calculated. Again,
  this can be limited to be no more than a maximum value ptfnumax by turning on constraint
  equation no. 54 with iteration variable no. 95 (fptfnuc). The inboard and outboard shield
  thicknesses also help this constraint to be met. (Note that in this calculation the TF coil case
  surrounding the superconductor winding pack is ignored.) This constraint equation may also be
  used with blktmodel = 0.
• The helium concentration in the vacuum vessel at the end of the plant lifetime is calculated. This needs to be constrained for re-weldability purposes, and can be kept below a maximum value \( v_{\text{healv}} \) by turning on constraint equation no. 55 with iteration variable no. 96 \( (v_{\text{vhe}}) \).

• The blanket lifetime is calculated, assuming a maximum allowable level of neutron damage to its steel of 60 dpa (currently not adjustable). (For the \( \text{blktmodel} = 0 \) model, the allowable blanket fluence \( abktflnc \) in MW-years/m\(^2\) may be input.)

### 3.1.4 Shield

The shield reduces the neutron flux reaching the vacuum vessel, TF coils and beyond. This minimises the radiological impact of the neutrons, and their heating of the TF coils which, if superconducting, need to remain at liquid helium temperatures. As with the blanket the energy deposited in the coolant may or may not be used to produce electricity, depending on the value of the switch \( \text{iprimshld} \). The shield coolant fraction by volume is set via the input parameter \( vfshld \).

The inboard and outboard shield thicknesses (\( shldith \) and \( shldoth \), respectively) may be used as iteration variables.

### 3.1.5 Divertor

The divertor provides a means of removing plasma reaching the scrape-off layer. The principal outputs from the code are the divertor heat load, used to determine its lifetime, and its peak temperature. The divertor is cooled either by gaseous helium or by pressurised water.

Switch \( s\text{null} \) controls the overall plasma configuration. Setting \( s\text{null} = 0 \) corresponds to an up-down symmetric, double null configuration, while \( s\text{null} = 1 \) (the default) assumes a single null plasma with the divertor at the bottom of the machine. The vertical build (see Figure 3.3) and PF coil current scaling algorithms take the value of this switch into account, although not the plasma geometry at present.

The Harrison-Kukushkin-Hotston divertor model \[16\] developed for ITER is available, but is unlikely to be relevant for a reactor.

### 3.1.6 Toroidal field coils

The toroidal field (TF) coils can be either resistive or superconducting. Switch \( \text{itfsup} \) should be set to 1 for superconducting coils, or 0 for purely copper coils. In the superconductor model, the \( \text{CICC} \) (Conductor In Cable Conduit) structure shown in Figure 3.6 is assumed, and the coils are cooled using a liquid helium cryogenic system.

The steel radial plates shown in the figure help in the winding process and also provide extra support against tangential stresses. Iteration variable no. 101 \( (prp) \) is the ratio of the total radial plate plus steel cap cross-sectional area within the winding pack to the total winding pack cross-sectional area; from this, the half-thickness \( trp \) is calculated.

The outboard leg of the TF coil is assumed to be the same width in the toroidal direction as the outside edge of the inboard leg. In the radial direction, for resistive TF coils the input parameter \( tfootfi \) gives the ratio of the outboard leg thickness to the inboard leg thickness \( tfcth \); for superconducting coils the outboard thickness is set equal to the inboard thickness.

Each TF coil is defined in the \( (R, Z) \) plane by a straight section and four elliptical arcs. Because of the finite number of TF coils used in a tokamak (18 for ITER), the toroidal field has a ripple introduced
Figure 3.6: Schematic diagram of the cross-section of the inboard leg of a superconducting TF coil, showing the CICC (Conductor In Cable Conduit) construction. The winding pack contains many turns of cable conduit. The cable space contains the superconducting filaments, and circulating liquid helium coolant. The variables shown in red may be changed by the user, and those in italics may be chosen as iteration variables.
into it, the amplitude of which can be limited to a few percent (given by input parameter \( \text{ripmax} \), default value 1%) by the code by adjusting the outboard gap thickness (labelled \( \text{gapsto} \) in Figures 3.1 and 3.2).

Among the TF coil parameters calculated by the code are the maximum allowable current density, the stresses on the structure, the energy stored and the magnetic field produced by the coils.

The following options are available within the superconducting TF coil model (\( \text{itfsup} = 1 \)).

### 3.1.6.1 Superconducting materials

Switch \( \text{isumattf} \) specifies which superconducting material is to be used:

- \( \text{isumattf} = 1 \): Nb\(_3\)Sn superconductor, ITER critical surface parameterization \([33]\), standard critical values
- \( \text{isumattf} = 2 \): Bi-2212 high temperature superconductor
- \( \text{isumattf} = 3 \): NbTi superconductor
- \( \text{isumattf} = 4 \): Nb\(_3\)Sn superconductor, ITER critical surface parameterization \([33]\), user-defined critical parameters

The fraction of copper present in the superconducting filaments is given by the value of variable \( f\text{cutfsu} \) (iteration variable no. 59).

For \( \text{isumattf} = 2 \), a technology adjustment factor \( f\text{hts} \) may be used to modify the critical current density fit for the Bi-2212 superconductor, to describe the level of technology assumed (i.e. to account for stress, fatigue, radiation, AC losses, joints or manufacturing variations). The default value for \( f\text{hts} \) is 0.5 (a value of 1.0 would be very optimistic).

For \( \text{isumattf} = 4 \), important superconductor properties may be input by the user as follows: the upper critical field at zero temperature and strain is set using input parameter \( \text{bcritsc} \), and the critical temperature at zero field and strain is set using input parameter \( \text{tcritsc} \).

### 3.1.6.2 Current density limits

The current in the TF coils must be sufficient to produce the required toroidal field at the centre of the plasma. In tokamaks, the field falls off at a rate \( 1/R \), with the peak value occurring at the outer edge of the inboard portion of the TF coil winding pack (\( R_{\text{max TF}} = r\text{bmax} \)). The maximum TF coil current depends on the field it produces and the allowable current density.

Three constraints are relevant to the operating current density \( J_{op} \) in the (superconducting) TF coils.

- To ensure that \( J_{op} \) does not exceed the critical value \( J_{crit} \), constraint equation no. 33 should be turned on with iteration variable no. 50 (\( f\text{iooic} \)).
- To ensure that \( J_{op} \) does not exceed the current density protection limit, constraint equation no. 35 should be turned on with iteration variable no. 53 (\( f\text{jprot} \)).
- The critical current density \( J_{crit} \) falls with the temperature of the superconductor. The temperature margin \( \Delta T \) is the difference between the temperature at which \( J_{crit} \) would be equal to \( J_{op} \) and the operating temperature. The minimum allowed \( \Delta T \) can be set using input parameter \( \text{tmargmin} \) together with constraint equation no. 36 and iteration variable no. 54 (\( f\text{tmargtf} \)).
Note that if the temperature margin is positive, \( J_{\text{op}} \) is guaranteed to be lower than \( J_{\text{crit}} \), and so constraints 33 and 36 need not both be turned on (in fact, it is recommended that only one of these two constraints is activated in any given run).

### 3.1.6.3 Stress model

Switch \( \text{tfc\_model} \) controls whether a simple stress model (\( \text{tfc\_model} = 0 \), suitable for solid copper TF coils) or a more complex stress model (\( \text{tfc\_model} = 1 \)) should be used. If \( \text{tfc\_model} = 1 \), a two-layer stress model [35] developed by CCFE is used.

To enforce the stress limits calculated using either of these models, constraint equation no. 31 (case stress) and/or constraint equation no. 32 (conduit stress) should be turned on with iteration variables no. 48 (\( \text{fstr\_case} \)) and/or no. 49 (\( \text{fstr\_cond} \)), respectively. The stress limit is set using input parameter \( \text{alstr\_tcf} \).

### 3.1.7 Poloidal field coils

The poloidal field (PF) coils are used initially to cancel the vertical field produced at the centre of the plasma by the central solenoid (Section 3.1.8) during start-up, and then to maintain the plasma position and shape during the flat-top period.

#### 3.1.7.1 PF coil positions

The positions and sizes of the PF coils are partly input, and partly calculated after consideration of the required currents and allowable current density.

The PF coil locations are controlled using a set of switches stored in array \( \text{ipf\_loc} \) (see Figure 3.1), and are calculated in routine \( \text{PFCOIL} \). The coils are (usually) organised into groups containing two PF coils placed symmetrically above and below the midplane, and each group \( j \) has an element \( \text{ipf\_loc}(j) \) assigned to it. Input parameter \( \text{ngrp} \) should be set to the number of groups, and \( \text{ncls}(j) \) should be assigned the number of coils in each group — which should be 2 in each case.

In the following, all variables are defined in the variable descriptor file \( \text{vardes.html} \). The values for \( \text{rpf1}, \text{rpf2}, \text{zref}(j) \) and \( \text{routr} \) should be adjusted by the user to locate the PF coils accurately.

The three possible values of \( \text{ipf\_loc}(j) \) correspond to the following PF coil positions: (Redo taking into account snlm and other recent changes e.g. rclsnorm)

\[
\text{ipf\_loc}(j) = 1: \text{PF coils are placed above the central solenoid (one group only)}; \\
R = \text{rohc} + \text{rpf1} \\
Z = \pm(\text{hmax} \ast \text{ohhghf} + 0.1 + 0.5 \ast (\text{hmax} \ast (1 - \text{ohhghf}) + \text{tfcth} + 0.1))
\]

\[
\text{ipf\_loc}(j) = 2: \text{PF coils are placed above the TF coils (one group only)}; \\
R = \text{rmajor} + \text{rpf2} \ast \text{triang} \ast \text{rminor} \\
Z = \pm(\text{hmax} + \text{tfcth} + 0.86)
\]

\[
\text{ipf\_loc}(j) = 3: \text{PF coils are placed radially outside the TF coils (any number of groups (?))}; \\
R = \text{rtot} + \text{tfthko}/2 + \text{routr} \\
Z = \pm(\text{rminor} \ast \text{zref}(j))
\]

The void fraction (for coolant) in each coil \( i \)'s winding pack is given by \( \text{vf}(i) \).
3.1.7.2 PF coil currents

The peak current per turn, \( cptd_{\text{in}}(i) \), and the winding pack peak current density \( rj_{\text{conpf}}(i) \) in each PF coil \( i \) are inputs. The PF coil currents vary as a function of time during the tokamak operation as indicated in Figure 3.7. They contribute part of the flux swing necessary to maintain the plasma current (see Section 3.1.8).

![Figure 3.7: Plot showing schematically the current waveforms for the plasma, a typical PF coil, and the central solenoid. Note that the currents in some of the PF coils may be the opposite sign to that shown, and the central solenoid current may remain positive during the \( I_p \) ramp-up period, although it will pass through zero during the burn phase.]

3.1.7.3 PF coil material

The PF coils can be either resistive or superconducting. This is determined from the value of \( ipf_{\text{res}} \). If \( ipf_{\text{res}} = 0 \), the PF coils and the central solenoid are assumed to be superconducting. If \( ipf_{\text{res}} = 1 \), they are assumed to be resistive, with their resistivity given by the value of variable \( pfcl_{\text{res}} \).

If \( ipf_{\text{res}} = 0 \), switch \( isumat_{\text{pf}} \) specifies which superconducting material is to be used for the PF coils. The values for \( isumat_{\text{pf}} \) are used in the same way as switch \( isumattf \) is for the TF coils (see Section 3.1.6.1).

The fraction of copper present in the superconducting filaments is given by the value of variable \( fcup_{\text{fsu}} \).

If the PF coils are superconducting, a steel case is assumed to surround the current-carrying winding pack to take the hoop stress. Its cross-sectional area is determined by the \( J \times B \) hoop force on the
coil divided by the allowable hoop stress, given by input parameter \( \text{sigpfcalw} \). The input parameter \( \text{sigpfcf} \) provides a scale factor (default is 0.666) to adjust the hoop force if required, to indicate what proportion of the force is supported by the case.

3.1.8 Central solenoid

Formerly known as the ohmic heating (OH) coil, the central solenoid is a PF coil used during start-up and during the burn phase to create and maintain the plasma current by inductive means. Swinging (changing) the current through the central solenoid causes a change in the flux linked to the plasma region, inducing a current in it. \text{PROCESS} calculates the amount of flux required to produce the plasma current, and also the amount actually available. The code measures the magnetic flux in units of Volt-seconds (= Webers).

Switch \( \text{iohcl} \) controls whether a central solenoid is present. A value of 1 denotes that this coil is present, and should be assigned a non-zero thickness \( \text{ohcth} \). A value of \( \text{iohcl} = 0 \) denotes that no central solenoid is present, in which case the thickness \( \text{ohcth} \) should be set to zero. No PF coils should be located at positions defined by \( \text{ipfloc(j)} = 1 \) if no central solenoid is present.

The central solenoid can be either resistive or superconducting (controlled via switch \( \text{ipfres} \) as for the other PF coils), and if superconducting, switch \( \text{isumato} \) determines the superconducting material to use — its value is used like \( \text{isumattf} \) and \( \text{isumatpf} \). The fraction of copper present in the superconducting filaments is given by the value of variable \( \text{fcuoohsu} \).

If the central solenoid is superconducting, the coil contains steel for strength. The cross-sectional area of steel is determined by the \( J \times B \) hoop force on the coil divided by the allowable hoop stress, given by the input \( \text{alstroh} \). A steel thickness is given in the output, which would be the thickness of a steel case of the same cross-sectional area if it simply surrounded the conducting region.

3.1.8.1 Current density inputs and limits

The (absolute value of the) central solenoid current density at the end-of-flat-top (‘EOF’), \( \text{coheof} \) is specified by the user, and can be used as an iteration variable (no. 37). The current density at the beginning-of-pulse (‘BOP’ — see Figure [3.7]) is specified as a (positive) fraction of \( \text{coheof} \) using \( \text{fcohbop} \) (iteration variable no. 41). The current density in the CS at all other times is calculated by taking account of the flux swing necessary to initiate and maintain the plasma current. The positive or negative sign of the current at each time is calculated automatically.

The current density in the central solenoid can be limited at the BOP and at the EOF. To limit the current density at the BOP, constraint equation no. 27 should be turned on with iteration variable no. 39 (\( \text{fjohc0} \)). To limit the current density at the EOF, constraint equation no. 26 should be turned on with iteration variable no. 38 (\( \text{fjohc} \)).

As for the TF coils, the critical current density \( J_{\text{crit}} \) falls with the temperature of the superconductor. The temperature margin \( \Delta T \) is the difference between the temperature at which \( J_{\text{crit}} \) would be equal to \( J_{\text{op}} \) and the operating temperature. The minimum allowed \( \Delta T \) can be set using input parameter \( \text{tmargmin} \) together with constraint equation no. 60 and iteration variable no. 106 (\( \text{ftmargoh} \)).

Note that if the temperature margin is positive, \( J_{\text{op}} \) is guaranteed to be lower than \( J_{\text{crit}} \), and so constraints 26, 27 and 60 need not all be turned on (in fact, it is recommended that EITHER the latter constraint, OR the former two constraints, is/are activated in any given run).
3.1.8.2 Plasma current ramp-up time

In the steady-state power plant scenario ($l_{\text{pulse}} \neq 1$ — see Section 3.3), the length of time taken for the central solenoid current to (possibly) reverse (which is equal to the plasma current ramp-up time — see Figure 3.7) is determined from the value of switch $t_{\text{tohsin}}$. If $t_{\text{tohsin}} = 0$, then the plasma current ramp-up time $t_{\text{tohs}}$ in seconds is given by $t_{\text{tohs}} = I_p/0.5$, where $I_p$ is the plasma current in MA. Furthermore, the PF coil ramp time $t_{\text{ramp}}$ and shutdown time $t_{\text{qnc}}$ are set equal to $t_{\text{tohs}}$. If $t_{\text{tohsin}} \neq 0$, the plasma current ramp-up time $t_{\text{tohs}} = t_{\text{tohsin}}$, and the PF coil ramp and shutdown times are input parameters.

If, however, a pulsed power plant is being modelled ($l_{\text{pulse}} = 1$), the plasma current ramp-up time $t_{\text{tohs}}$ is either an input parameter, or it can be iterated by using iteration variable 65. The ramp and shutdown times in the pulsed case are always set equal to $t_{\text{tohs}}$. To ensure that the plasma current ramp rate during start-up is prevented from being too high, as governed by the requirement to maintain plasma stability by ensuring that the induced current has time to diffuse into the body of the plasma, constraint equation no. 41 should be turned on with iteration variable no. 66 ($f_{\text{tohs}}$).

3.1.9 Auxiliary power systems: heating and current drive

3.1.9.1 Current Drive

The use of inductive current drive leads to pulsed plant operation because of the limited flux swing that can be achieved using the central solenoid. This poses problems due to the fact that fatigue failures may result, and there would also be a need for thermal storage to maintain output of electricity between pulses, and supply power for starting a new pulse. However, the plasma current can also be produced and maintained (partially or wholly) using non-inductive means which, in principle, removes this restriction. PROCESS contains a number of auxiliary current drive schemes, including various RF methods (Lower Hybrid, Electron Cyclotron, and Ion Cyclotron (Fast Wave) current drives) and also Neutral Beam current drive systems. The code calculates the efficiency and the resulting power requirements of the chosen system.

The fraction of the required plasma current to be produced by non-inductive means, $f_{\text{vsbrnni}}$, should be set, and flag $\text{irfcd}$ should be set to 0 for purely inductive scenarios, or 1 otherwise. The current drive efficiency model to be used in this latter case is defined by the value of switch $\text{iefrf}$:-

\begin{verbatim}
    iefrf = 1 : Fenstermacher Lower Hybrid model
    iefrf = 2 : Ion cyclotron model [16]
    iefrf = 3 : Fenstermacher electron cyclotron resonance model
    iefrf = 4 : Ehst Lower Hybrid model
    iefrf = 5 : ITER neutral beam model [16, 17]
    iefrf = 6 : Culham Lower Hybrid model [17]
    iefrf = 7 : Culham electron cyclotron model [17]
    iefrf = 8 : Culham neutral beam model [17]
    iefrf = 9 : Oscillating Field current drive (RFPs only — see Section 3.6.1.5)
\end{verbatim}
(Note that, at present, the neutral beam models do not include the effect of an edge transport barrier (pedestal) in the plasma profile.)

It is sometimes useful to adjust artificially the current drive efficiency values produced by these routines. This can be achieved by setting the scaling coefficient \( f_{\text{effcd}} \). The wall plug to plasma efficiencies can also be adjusted, by changing the relevant variable (\( \eta_{\text{ech}}, \eta_{\text{alh}}, \eta_{\text{nbi}} \) or \( \eta_{\text{of}} \)).

### 3.1.9.2 Plasma heating

In addition to current drive, some auxiliary power can be used purely to heat the plasma. The value of input parameter \( p_{\text{heat}} \) determines the amount of auxiliary heating power (in Watts) to be applied to the plasma. This variable may be used as an iteration variable (no. 11).

### 3.1.9.3 Neutral beam access

If present, a neutral beam injection system needs sufficient space between the TF coils to be able to intercept the plasma tangentially. The major radius \( r_{\text{tanbeam}} \) at which the centreline of the beam is tangential to the toroidal direction is user-defined using input parameter \( f_{\text{rbeam}} \), which is the ratio of \( r_{\text{tanbeam}} \) to the plasma major radius \( r_{\text{major}} \). The maximum possible tangency radius \( r_{\text{tanmax}} \) is determined by the geometry of the TF coils — see Figure 3.8 and this can be enforced using constraint equation no. 20 with iteration variable no. 33 (\( f_{\text{portsz}} \)). The thickness of the beam duct walls may be set using input parameter \( \text{nbshield} \).

### 3.1.9.4 Neutral beam losses

Input parameter \( f_{\text{orbitloss}} \) can be used to specify the fraction of the net injected neutral beam power that is lost between the beam particles’ ionisation and thermalisation (known as the first orbit loss). This quantity cannot easily be calculated as it depends on the field ripple and other three-dimensional effects. The power lost is assumed to be absorbed by the first wall.

The power in the beam atoms that are not ionised as they pass through the plasma (shine-through) is calculated by the code. There are two constraint equations that can be used to control the beam penetration and deposition, as follows:

- It is necessary to use a beam energy that simultaneously gives adequate penetration of the beam to the centre of the plasma and tolerable shine-through of the beam on the wall after the beam has traversed the plasma. The number of exponential decay lengths, \( \tau \), for the beam power to fall before it reaches the plasma centre should be in the region of \( \sim 4–6 \) \[17\] Section 4.3.2]. Constraint equation no. 14 may be used to force \( \tau \) to be equal to the value given by input parameter \( t_{\text{beamin}} \), and is therefore in effect a beam energy consistency equation.

- Alternatively, constraint equation no. 59 with iteration variable no. 105 (\( f_{\text{nbshinef}} \)) may be used to ensure that the beam power fraction emerging from the plasma is no more than the value given by input parameter \( \text{nbshinefmax} \).

It is recommended that **only one** of these two constraint equations is used during a run.
Figure 3.8: Top-down schematic view of the neutral beam access geometry. The beam with the maximum possible tangency radius is shown here.
3.1.9.5 Ignited plasma

Switch \texttt{ignite} can be used to denote whether the plasma is ignited, i.e. fully self-sustaining without the need for any injected auxiliary power during the burn. If \texttt{ignite} = 1, the calculated injected power does not contribute to the plasma power balance (Section 3.1.2.11), although the cost of the auxiliary power system is taken into account (the system is then assumed to be required to provide heating etc. during the plasma start-up phase only — use \texttt{pheat} to indicate the power requirement). If \texttt{ignite} = 0, the plasma is not ignited, and the auxiliary power is taken into account in the plasma power balance during the burn phase. Also, constraint equation no. 28 can be turned on to enforce the fusion gain \( Q \) to be at least \( \texttt{bigqmin} \).

3.1.10 Structural components

Structural components are required to provide support for the fusion power core systems against gravity and the magnetic forces that will be encountered during operation. The required structural masses and their costs are calculated.

3.1.11 Cryostat and vacuum system

The internal vacuum vessel provides a toroidal evacuated chamber containing the plasma, first wall, blanket and shield, and the space between this item and the external cylindrical cryostat encloses those components that need to operate at liquid helium temperatures. These include any superconducting (TF or PF) coils and the inter-coil structure. \texttt{PROCESS} calculates the cryogenic power load and the resulting heat exchanger requirements.

The vertical distance \( h \) between the uppermost PF coil and the external cryostat lid may be adjusted by changing the value of input parameter \texttt{clhsf}; a scaling based on ITER is used:

\[
    h = \texttt{clhsf} \left( \frac{2 \times \texttt{rdewex}}{28.440} \right)
\]

(3.22)

The vacuum system is used for four different processes. Firstly, before plasma operations the chamber must be evacuated to remove outgassed impurities from the structure. Secondly, the chamber must be re-evacuated between burn operations. Thirdly, helium ash must be removed to prevent it from diluting the fuel. Finally, deuterium and tritium is removed on a steady state basis. \texttt{PROCESS} calculates the parameters of a vacuum system that satisfy all four requirements, with the option of either turbo pumps or cryo pumps being used.

Switch \texttt{ntype} controls whether a turbopump (\texttt{ntype} = 0) or a cryopump (\texttt{ntype} = 1) is used in the vacuum system.

3.1.12 Power conversion and heat dissipation systems

The \texttt{PROCESS} power plant takes into account all the systems required to perform the necessary conversion of fusion power to electricity, from the coolant systems in the plant components to the heat exchangers and turbines.

Figure 3.9 shows the overall power transfer mechanisms within the power plant outside of the plasma. The efficiency of the primary coolant pumps in converting electrical to mechanical power is given by input parameter \texttt{etahtp} in all the calculations described below.
3.1.12.1 Divertor

All of the charged particle transport power leaving the plasma (excluding the $1-falpha$ portion of the alpha power that escapes directly to the first wall — Section 3.1.2.11) is assumed to be absorbed in the divertor, along with a proportion $f_{div}$ of the radiation power and the neutron power. Switch $iprim_{div}$ may be used to specify whether the thermal power deposited in the divertor becomes high-grade thermal power ($iprim_{div} = 1$) or low-grade waste heat (see Figure 3.9).

3.1.12.2 First wall

The remaining photon power (i.e. the fraction not incident upon the divertor or lost through holes or the heating / current drive apparatus) is assumed to be absorbed by the first wall. Power due to ions derived from the neutral beams but lost before being thermalised (known as "first orbit loss"), and from the fast alpha particles lost before being thermalised, also contribute to the total thermal power absorbed by the first wall.

3.1.12.3 Power conversion cycle

Figure 3.9 summarises the power conversion mechanisms.

The high-grade, i.e. useable thermal power (less any thermal power required to produce hydrogen in a hydrogen production plant — see Section 3.4) is used to produce steam to turn the turbines, thus generating the plant’s gross electrical power, with an efficiency given by quantity $\eta_{th}$. The remainder is dumped to the environment through the condenser. All of the low-grade heat is dumped to the environment.
**primary_pumping** : This switch controls the calculation of the mechanical pumping power required for the primary coolant.

If `primary_pumping = 0`, the user sets mechanical pumping power directly.

If `primary_pumping = 1`, the user sets mechanical pumping power as a fraction of thermal power removed by coolant.

If `primary_pumping = 1`, the mechanical pumping power is calculated, as follows.

User inputs for the coolant outlet temperature (which may be used as an iteration variable), the coolant channel diameter, and the segmentation of the blanket are used. The peak temperature in the first wall material (underneath the armour) is derived. The user can apply an upper limit to this temperature, and if this constraint is used then it is strongly recommended to set the length of a first wall pipe (`fw_channel_length`) as an iteration variable. The Gnielinski correlation is used to determine the heat transfer in the channel. The stresses in the first wall are not currently taken into account.

The mechanical pumping power required for the first wall and breeder zone is calculated using the Darcy friction factor, estimated from the Haaland equation, an approximation to the ColebrookWhite equation. (If you consider that the calculated value for the pumping power is too low, then you can add an additional electric power requirement using `baseel`. Do not use `htpmw_min` as this prevents the optimisation of the first wall.) The inlet and outlet temperatures of the first wall and blanket can be different. The isentropic efficiency of the first wall and blanket coolant pumps (enthalpy increase in the fluid for isentropic compression divided by the mechanical power used) is specified by the parameter `etaiso`. Note that the mechanical pumping powers for the shield and divertor are still calculated using the simplified method (a fixed fraction of the heat transported).

**secondary_cycle** : This switch controls how the calculation of the plant’s thermal to electric conversion efficiency (the secondary cycle thermal efficiency) proceeds. See also Section 3.1.12, which describes the flow of power through the fusion power core.

If `secondary_cycle = 0`, the efficiency of the power generation cycle is set to a single value obtained from previous cycle modelling studies. The heat deposited in the toroidal field coils divertor coolant is assumed to be at such low temperature that it cannot be used for power generation and is dumped to the environment.

If `secondary_cycle = 1`, the efficiency of the power generation cycle is set as above, but the divertor heat is used for electricity generation.

In the remaining options (`secondary_cycle = 1, 2 or 3`), the heat deposited in the divertor coolant is used for power generation.

If `secondary_cycle = 2`, the efficiency of the power generation cycle is input by the user.

If `secondary_cycle = 3`, a steam Rankine cycle is assumed. The secondary cycle thermal efficiency (`etath`) is calculated from the coolant outlet temperature using simple relations between temperature and efficiency [32]:

\[
\eta = -2.0219 + 0.3720 \ln(T) \quad \text{(water coolant; saturated steam Rankine cycle)} \quad (3.23)
\]

\[
\eta = -0.8002 + 0.1802 \ln(T) \quad \text{(helium coolant; superheated steam Rankine cycle)} \quad (3.24)
\]

If `secondary_cycle = 4`, a supercritical CO₂ Brayton cycle is assumed. The secondary cycle thermal efficiency (`etath`) is calculated from the coolant outlet temperature using simple relations between temperature and efficiency from [32]:

\[
\eta = -2.5043 + 0.4347 \log(T) \quad \text{(water or helium coolant)} \quad (3.25)
\]
In the above three equations, $T$ is the temperature of the (secondary) coolant at the inlet to the turbine, assumed to be a fixed margin below the outlet temperature of the primary coolant.

The electrical power required to operate the power plant itself is the so-called recirculating electric power. Any surplus is exported to the electricity grid as net electric power.

The recirculating power comprises the electrical power required to run all of the associated electrical systems surrounding the fusion power core, plus the on-site building services, offices, etc., as shown in Figure 3.9. Of these, the cryogenic plant power includes the power required to cool the TF coils from the neutron power absorbed by the coils, the PF coils (as defined by the ratio of the total PF coil stored energy to the fusion power pulse time $t_{\text{pulse}}$), and other ‘cold’ components.

### 3.1.13 Buildings

The volume and ground area of all the various buildings on a power plant site are included in the PROCESS calculations for the benefit of the costing algorithms.

### 3.2 Spherical Tokamak Model

PROCESS has the ability to perform studies on tokamaks in the low aspect ratio regime (major radius $\leq 2 \times$ minor radius). The physics and engineering issues associated with these machines are somewhat different from those of conventional aspect ratio, and this is reflected by the following special models in PROCESS.

1. The inboard build of a spherical tokamak (ST) is very different from that in a conventional tokamak. There is no inboard blanket (and possibly no inboard shield), and the inboard TF coil legs are replaced by a single centrepost. The radial build is altered so that, starting from the centreline ($R = 0$), the component order is: TF coil, gap, central solenoid, vacuum vessel, and then continuing as in Figure 3.1 (a D-shaped cross-section is assumed for the first wall, blanket, shield and vacuum vessel).

2. Spherical tokamaks have resistive TF coils that combine into a single centrepost at the centre of the machine. The centrepost is constructed from copper (as are the outboard TF coil sections), and is tapered lengthways so that it is narrowest at the midplane of the device. Routine CNTRPST calculates various parameters relevant to the centrepost, including the pump pressure, maximum temperature and pipe radius, and these may be limited using constraint equations 43 to 46 if required:
   - Equation 43 is a consistency equation for the average centrepost temperature.
   - Equation 44 can be used to limit the peak centrepost temperature to a maximum value ($p_{\text{tempalw}}$) using iteration variable no. 68 ($f_{\text{ptemp}}$).
   - Equation 45 can be used to force a lower limit to the edge safety factor $q_{\text{lim}}$ (see below), using iteration variable no. 71 ($f_q$).
   - Equation 46 can be used to apply an upper limit to the ratio of plasma current to TF coil (“rod”) current, using iteration variable no. 72 ($f_{\text{pir}}$).

3. A gaseous divertor model is used, and a simple divertor heat load calculation is employed, rather than the more complex divertor model assumed for conventional aspect ratio tokamaks.

4. A simple PF coil current scaling algorithm is available for use with the ST option.
5. The plasma shaping terms (elongation and triangularity) can be calculated directly given the aspect ratio, using \( \text{ishape} = 1 \) (see Section 3.1.2.1). This setting also scales the lower limit of the edge safety factor, for use with constraint equation no. 45:

\[
q_{\text{lim}} = 3(1 + 2.6 \times \epsilon^{2.8})
\]

(3.26)

where \( \epsilon = \frac{a}{R} \).

6. Among the physics models that differ from those relevant to conventional aspect ratio machines are (i) the plasma poloidal field \( B_{\text{pol}} \), (ii) the bootstrap current fraction, (iii) the beta limit, and (iv) the neutron heating of the centrepot [2].

### 3.2.1 Spherical tokamak switches

Switch \( \text{itart} \) provides overall control of the ST switches within the code, and subroutine \( \text{CHECK} \) ensures that no conflicting values are inadvertently set by the user in the input file. Table 3.3 summarises the switch values relevant to each aspect ratio regime.

<table>
<thead>
<tr>
<th>switch</th>
<th>conventional aspect ratio</th>
<th>low aspect ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \text{ishape} ) (Section 3.1.2.1)</td>
<td>( \text{itart} = 0 ) 0, 2</td>
<td>( \text{itart} = 1 ) 0, 1</td>
</tr>
<tr>
<td>( \text{ibss} ) (Section 3.1.2.12)</td>
<td>( \text{itart} = 0 ) 1, 2, 3</td>
<td>( \text{itart} = 1 ) 2, 3</td>
</tr>
<tr>
<td>( \text{icurr} ) (Section 3.1.2.8)</td>
<td>( \text{itart} = 0 ) 1, 3, 4, 5, 6, 7</td>
<td>( \text{itart} = 1 ) 2</td>
</tr>
<tr>
<td>( \text{itfsup} ) (Section 3.1.6)</td>
<td>( \text{itart} = 0 ) 0, 1</td>
<td>( \text{itart} = 1 ) 0</td>
</tr>
</tbody>
</table>

Table 3.3: Summary of the switch values in PROCESS that relate to conventional aspect ratio and low aspect ratio machines.

### 3.3 Pulsed Plant Operation

If the plasma current is partially or entirely driven by electromagnetic induction, it is necessary to operate the plant in a pulsed manner as the current swing in the central solenoid coils cannot be continued indefinitely. PROCESS can perform a number of calculations relevant to a pulsed power plant, as detailed below.

Switch \( \text{lpulse} \) determines whether the power plant is assumed to be based on steady-state (\( \text{lpulse} = 0 \)) or pulsed (\( \text{lpulse} = 1 \)) operation.

#### 3.3.1 Thermal cycling package

Update... This performs calculations on the first wall of the machine. Evaluation of the mechanical and thermal stresses on this component lead to a measure of the maximum number of cycles to which the first wall can be subjected, and hence to the minimum allowable length of each reactor cycle for a specified first wall lifetime. The cycle time can be constrained to be at least the minimum value by turning on constraint equation no. 42 with iteration variable no. 67 (\( \text{ftcycl} \)).

The thickness of the first wall is constrained to lie within lower and upper bounds, which ensures that it can withstand the internal coolant pressure and the peak temperature and neutron fluence.
Switch \textit{itcycl} activates the desired model for the first wall axial stress calculations. If \textit{itcycl} = 1 (the default), the wall is fully constrained axially, and no bending can occur. If \textit{itcycl} = 2, there is no constraint on the axial motion, but no bending can occur. Finally, if \textit{itcycl} = 3, again there is no axial constraint, and bending is allowed to occur.

\subsection*{3.3.2 First wall coolant temperature rise limit}

The rise in temperature of the first wall coolant can be limited to be no more than the value of \textit{dtmpmx} by turning on constraint equation no. 38 with iteration variable no. 62 (\textit{fdtmp}).

\subsection*{3.3.3 First wall peak temperature limit}

The maximum first wall temperature can be limited to be no more than the value of variable \textit{tpkmax} by turning on constraint equation no. 39 with iteration variable no. 63 (\textit{ftpeak}).

\subsection*{3.3.4 Start-up power requirements}

The minimum auxiliary power required during the start-up (ignition) phase is calculated on the basis of a POPCON analysis. Ignition is accessed via the so-called Cordey Pass (the path in plasma density–temperature space which minimises the power requirement) and the code ensures that there is sufficient auxiliary power to accommodate this. In fact, this calculation is very CPU-intensive, so the relevant routine is not called at present. In practice, the auxiliary power tends to exceed the minimum allowable value anyway, without any need to constrain it to do so.

The auxiliary power reaching the plasma can be forced to be more than the minimum allowable value \textit{auxmin} by turning on constraint equation no. 40 with iteration variable no. 64 (\textit{fauxmn}). The value of \textit{auxmin} is determined by the code if the start-up model is activated, otherwise it may be initialised via the input file.

\subsection*{3.3.5 Plasma current ramp-up time}

This calculation ensures that the plasma current ramp rate during start-up is prevented from being too high, as governed by the requirement to maintain plasma stability in \(l_i - q_\psi\) space (see Section 3.1.8.2).

\subsection*{3.3.6 Burn time}

The length of the burn time is calculated from the surplus volt-seconds available from the OH/PF coil system during the plasma burn phase, after the flux required during the plasma start-up is taken into account. A minimum burn time can be enforced via constraint equation no. 13 and iteration variable no. 21 (\textit{ftburn}).

\subsection*{3.3.7 Thermal storage}

During every cycle there is a period when no fusion power is produced. The net electric output from the plant must, however, be maintained, and this is achieved using thermal storage. There are three types of thermal storage available within \textsc{process}, and the value of switch \textit{istore} determines which is to be used. If \textit{istore} = 1 (the default), option 1 of Ref. [35] is assumed, which utilises the thermal storage inherent in the machine’s steam cycle equipment. This should be used if the machine down
time is less than 100 seconds. If $\text{store} = 2$, option 2 of Ref. \[35\] is assumed, which uses the same method as before, but augments it with an additional boiler. This may be used for machine down times of up to 300 seconds. Finally, if $\text{store} = 3$, a large stainless steel block acts as the thermal storage medium.

### 3.4 Hydrogen Production Facility

Fusion power plants have been mooted as a means of producing hydrogen for use in fuel cells for cars, for instance. \textsc{Process} includes options to enable the plant to produce hydrogen using a number of different processes.

To include the production of hydrogen by the power plant, it is necessary to set the switch $\text{ihplant}$, as follows:

- $\text{ihplant} = 0$: No hydrogen production (default)
- $\text{ihplant} = 1$: Hydrogen production by low temperature electrolysis
- $\text{ihplant} = 2$: Hydrogen production by endothermic high temperature electrolysis
- $\text{ihplant} = 3$: Hydrogen production by exothermic high temperature electrolysis
- $\text{ihplant} = 4$: Hydrogen production by thermo-chemical processes

Table 3.4 describes the additional options available for each of the types of hydrogen production given above. The different processes use either electrical power or thermal power directly, so the required inputs differ. Variable $\text{helecmw}$ (iteration variable no. 87) is the electrical power in MW required for hydrogen production, while $\text{hthermmw}$ (iteration variable no. 88) is the thermal power required. Note that $\text{hthermmw}$ must not be used as an iteration variable if $\text{ihplant} \neq 4$, as it will be calculated from the required electrical power instead. Similarly, $\text{helecmw}$ must not be used as an iteration variable if $\text{ihplant} = 4$.

<table>
<thead>
<tr>
<th>hydrogen plant option</th>
<th>$\text{helecmw}$</th>
<th>$\text{hthermmw}$</th>
<th>efficiency variable</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{ihplant}=1$</td>
<td>input</td>
<td>zero</td>
<td>$\eta_{\text{ih}}$</td>
</tr>
<tr>
<td>$\text{ihplant}=2$</td>
<td>input</td>
<td>calculated</td>
<td>$\eta_{\text{ht}}$</td>
</tr>
<tr>
<td>$\text{ihplant}=3$</td>
<td>input</td>
<td>calculated</td>
<td>$\eta_{\text{ht}}$</td>
</tr>
<tr>
<td>$\text{ihplant}=4$</td>
<td>zero</td>
<td>input</td>
<td>$\eta_{\text{ht}}$</td>
</tr>
</tbody>
</table>

Table 3.4: \textit{Summary of the variables in \textsc{Process} that relate to the different hydrogen plant processes.}

The efficiency variables given in Table 3.4 are all input parameters, and are the factors to be used to convert the value of $\text{helecmw}$ to the amount of hydrogen produced (in MW equivalent); these can be greater than unity in all cases except $\text{ihplant} = 4$.

### 3.5 Stellarator Model

The code has the ability to perform calculations based on the physics and engineering of a stellarator, which, although being a toroidal device, is radically different in a number of ways from a tokamak.

The model is largely based on W7-X and the HELIAS 5-B stellarator power plant design \[39\] (Figure 3.10) and related modelling that has been performed by IPP Greifswald \[40, 41, 42\].
To activate the stellarator coding, it is necessary to create a file `device.dat`, containing the single character 1 in the first row, in the working directory (see Section 4.4). This has the effect of setting the internally-used switch `istell = 1`. If the file is absent, or its first character is set to something other than 1, the stellarator model is not used, and `istell` is set to 0.

The stellarator model is largely contained within source file `stellarator.f90`. The following consistency equations (see Section 2.5) should be used without modification:

1: plasma beta consistency
2: global power balance
11: radial build consistency

Figure 3.10: Fusion power core of the HELIAS 5-B conceptual power plant design (from [39])

3.5.1 Stellarator physics

Much of the physics is identical to that for tokamaks, including the plasma composition and the fusion power calculations. However, some physics topics do differ between stellarators and tokamaks, as follows.
3.5.1.1 Plasma geometry

The plasma geometry model uses Fourier coefficients to represent the complex 3-D plasma shape typical of stellarators. A VMEC [13] calculation (or other equilibrium code that can provide the Fourier coefficients of the LCFS) must have been performed prior to running with PROCESS. The overall size of the plasma is scaled correctly for the required (mean) major and minor radii for the machine being modelled [14].

It is necessary to provide three input files that define the plasma shape:

1. Set `vmeceanfile` in the input file to the name of a file with the following format (the numbers given are those for the W7-X high mirror configuration); in practice only the effective major and minor radius values ($R_{\text{eff}}$ and $a_{\text{eff}}$, respectively) and the number of field periods are used:

<table>
<thead>
<tr>
<th>$R_{\text{eff}}$ [m]</th>
<th>$a_{\text{eff}}$ [m]</th>
<th>Aspect ratio</th>
<th>Plasma vol [m$^3$]</th>
<th>LCFS surf area [m$^2$]</th>
<th>field periods</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.486576</td>
<td>0.4942638</td>
<td>11.10050</td>
<td>26.45749</td>
<td>133.4281</td>
<td>5.0</td>
</tr>
</tbody>
</table>

2. Set `vmeceanrmnfile` in the input file to the name of a VMEC output file containing the calculated plasma boundary $R(m,n)$ Fourier coefficients, where $m = 0$ to 11 (rows) and $n = -12$ to 12 (columns).

3. Set `vmeceanzmnfile` in the input file to the name of a VMEC output file containing the calculated plasma boundary $Z(m,n)$ Fourier coefficients, where $m = 0$ to 11 (rows) and $n = -12$ to 12 (columns).

This method enables a wide range of potential plasma geometries to be modelled, if required. The plasma volume, surface area and mean cross-sectional area are the outputs from the plasma geometry model.

3.5.1.2 Absense of plasma current

Stellarators try to achieve zero plasma current in order to allow safe divertor operation, so no current scalings are required.

3.5.1.3 Beta limit

The beta limit is assumed to be 5%, based on 3-D MHD calculations [15]. To apply the beta limit, constraint equation no. 24 should be turned on with iteration variable no. 36 (`fbetatry`).

3.5.1.4 Density limit

The density limit relevant to stellarators has been proposed to be [16]

$$n_{\text{max}} = 0.25 \left( \frac{PB_0}{R_0 a_p^2} \right)^{\frac{1}{2}}$$  \hspace{1cm} (3.27)

where $n$ is the line-averaged electron density in units of $10^{20}$ m$^{-3}$, $P$ is the absorbed heating power (MW), $B_0$ is the on-axis field (T), $R_0$ is the major radius (m), and $a_p$ is the plasma minor radius (m). To enforce the density limit, turn on constraint equation no. 5 with iteration variable no. 9 (`fdene`).
3.5.1.5 $\tau_E$ scaling laws

Five confinement time scaling laws relevant to stellarators are present within PROCESS. The value of switch $isc$ determines which of these is used in the plasma energy balance calculation.

$$\tau_E \text{ (Large Helical Device \[46\]: } isc=21) = 0.17 R_0^{0.75} a_p^{2.5} n_2^{0.69} B_0^{0.84} P^{-0.58}$$

$$\tau_E \text{ (Gyro-reduced Bohm \[47\]: } isc=22) = 0.25 B_0^{0.8} n_2^{0.6} P^{-0.6} a_p^{2.4} R_0^{0.6}$$

$$\tau_E \text{ (Lackner-Gottardi \[48\]: } isc=23) = 0.17 R_0 a_p^{2.5} n_2^{0.6} B_0^{0.8} P^{-0.6} \bar{i}^{0.4}$$

$$\tau_E \text{ (ISS95 \[49\]: } isc=37) = 0.079 R_0^{0.65} a_p^{2.21} n_1^{0.51} B_0^{0.83} P^{-0.59} \bar{i}^{0.4}$$

$$\tau_E \text{ (ISS04 \[50\]: } isc=38) = 0.134 R_0^{0.64} a_p^{2.28} n_1^{0.54} B_0^{0.84} P^{-0.61} \bar{i}^{0.41}$$

Here, $\bar{i}$ is the rotational transform, which is equivalent to the reciprocal of the tokamak safety factor $q$.

3.5.1.6 Heating power options

Stellarators require no current drive, although provision for auxiliary heating does need to be present. The method by which auxiliary heating power is supplied is determined by the switch $isthtr$:

- $isthtr = 1$ : electron cyclotron resonance heating
- $isthtr = 2$ : lower hybrid heating
- $isthtr = 3$ : neutral beam injection

The value of variable $pheat$ determines the actual amount of auxiliary heating power (in Watts) to be applied to the plasma. This variable may be used as an iteration variable (no. 11). Switch $ignite$ may be used if necessary — see Section 3.1.9.5.

3.5.1.7 Divertor

Although the divertor has the same function in both stellarators and tokamaks, the envisaged concepts differ quite substantially. This is not only related to the different geometry and symmetries but also specifically to the magnetic configuration. While the inherent axisymmetry of a tokamak allows for poloidally-continuous single or double divertor configurations, the periodicity of helical advanced stellarators leads to multiple X-points with a corresponding chain of magnetic islands. This island structure may be exploited by carefully placing divertor plates along the stochastic field lines, naturally leading to a discontinuous divertor, with the individual plates being placed in a complex 3-D geometry; see Figure 3.11.

Rather than trying to describe the complex physics with a two-point scrape-off layer model as is used for tokamaks, the stellarator divertor model \[11\] is based on fundamental principles which relate the power crossing the separatrix with an effective wetted area on the divertor plates allowing the code to estimate the heat load delivered to the divertor. A basic island divertor model is used which assumes diffusive cross-field transport and high radiation at the X-point.

The radiated power fraction in the scrape-off layer is given by the input parameter $f_rad$. This is in contrast to the method used for tokamaks, in which the radiated power fraction is a calculated quantity.

A number of other input parameters may be used to modify the divertor model; see the variable descriptor file for more details.
3.5.2 Machine configuration

There are a large number of possible stellarator configurations. As stated earlier, the one chosen for the PROCESS model is based on the HELical Advanced Stellarator (HELIAS) concept, in which all the coils resemble distorted, non-planar TF coils — no helical coils or tokamak-like PF coils are present. The coil geometry is scaled from the HELIAS 5-B power plant design, which is based on a five field-period HELIAS configuration.

3.5.2.1 Machine build

Since a stellarator is inherently non-axisymmetric, the build of the PROCESS stellarator is defined in terms of the mean thicknesses of components.

The surface areas for the first wall, blanket and shield components are scaled linearly with their effective minor radius from the plasma surface area calculation (the area of a simple torus of circular cross-section is $4\pi R a$, hence the linear scaling with $a$). The input parameter $f_{\text{hole}}$ may be used to specify the hole fraction, to adjust the surface area to take account of ports and divertors, for instance. The volumes of the first wall etc. are simply given by the product of their surface area and their mean thickness.

In contrast to tokamaks, which in PROCESS are assumed to have a cylindrical external cryostat completely surrounding the fusion power core, the stellarator model assumes that the external cryostat (labelled as the outer vessel in Figure 3.10) is toroidal with a circular cross-section. Its cross-section is assumed to be centred at the mean plasma major radius.
All items external to the fusion power core (buildings, turbines, power conversion systems, etc.) remain unchanged.

3.5.2.2 Stellarator coils

The stellarator coil model \cite{42, 51} combines scaling aspects based on the HELIAS 5-B design in combination with analytic inductance and field calculations.

The fully three-dimensional shape of the coils is assumed to be fixed, but the sizes of the coils are scaled from the HELIAS 5-B values to the geometrical values for the machine being modelled using fundamental physics and engineering principles.

The stellarator coils are assumed to be superconducting — no resistive coil calculations are performed. The critical field at the superconductor is calculated using circular approximations for the coils in the inductance and field calculations, and the limit is enforced automatically. Available superconductors are Nb$_3$Sn ($\text{isumattf} = 1$) and NbTi ($\text{isumattf} = 3$).

The winding pack cross-section is rectangular for the stellarator coils, rather than the complicated two-step cross-section assumed for tokamaks. The coil thicknesses and most of the dimensions of the materials within the coil cross-section are outputs from the model, instead of being inputs as is the case for tokamaks; see the variable descriptor file for details. In addition, certain iteration variables ($\text{tfcth}$, no. 13; $\text{thkcas}$, no. 57; $\text{cpttf}$, no. 60 and $\text{tftort}$, no. 77) should not be turned on in the input file as they are calculated self-consistently; the code will stop with an error message if this is attempted.

3.6 Reversed Field Pinch Model

In addition to the tokamak and stellarator magnetic confinement devices, the code has the ability to perform calculations based on the physics and engineering of a reversed field pinch (RFP) device. This third type of toroidal magnetic device is superficially similar in design to a tokamak (so therefore shares many of the same components), but the magnetic field configuration differs.

The model used in PROCESS is largely based on the TITAN fusion power plant \cite{52, 53}. The following sections summarise its main features, where they differ from those for tokamaks.

To activate the RFP coding, it is necessary to create a file `device.dat`, containing the single character 2 in the first row, in the working directory (see Section 4.4). This has the effect of setting the internally-used switch $\text{irfp} = 1$. If the file is absent, or its first character is set to something other than 2, the RFP model is not used, and $\text{irfp}$ is set to 0.

3.6.1 RFP physics

The plasma in RFPs is circular in cross-section and is axisymmetric.

3.6.1.1 Beta limit

The poloidal beta is limited to a maximum value given by input parameter $\text{betpmx}$ (default = 0.19) using constraint equation 48 and iteration variable 79 ($\text{fbetap}$).
3.6.1.2 Density limit

No density limit is explicitly coded for RFPs, other than by simply constraining the upper bound of the electron density variable $d_{\text{ene}}$ (iteration variable 6).

3.6.1.3 $\tau_E$ scaling law

One confinement time scaling law relevant to RFPs is present within PROCESS. The value of switch $\text{isc}$ determines the scaling to be used in the plasma energy balance calculation.

$$\tau_E (\text{TITAN RFP} \ [52]: \ isc=25) = 0.05 a^2 I_p (\text{MA})$$

3.6.1.4 $F$-$\Theta$ plot

Much of the RFP physics is derived from the characteristic $F$-$\Theta$ plot, where $F = B_\theta (a)/\langle B_\phi \rangle$, $\Theta = B_\theta (a)/\langle B_\phi \rangle$, and $\langle B_\phi \rangle$ is the average value of the toroidal field over the plasma cross-section. Given a value of the pinch parameter $\Theta$, the corresponding value of the reversal parameter $F$ may be read from the $F$-$\Theta$ plot, and from these the plasma current and the current in the TF coils may be obtained using

$$B_\theta (a) = \frac{\mu_0 I_p}{2\pi a}$$
$$B_\phi (a) = \frac{\mu_0 I_{\text{TFC}}}{2\pi R}$$

(the second of these assumes that $B_\phi (a)$ is approximately equal to the vacuum toroidal field at the plasma centre).

The pinch parameter $\Theta$ is set using iteration variable no. 78: $rpfth$. The corresponding value of the reversal parameter $F$ ($rpf$) is calculated using routine $\text{FTTHETA}$. $F$ is constrained to be negative using constraint equation no. 49 with f-value $\text{frpf}$ (iteration variable no. 80).

3.6.1.5 Current drive

The RFP oscillating field current drive option is turned on by setting $\text{iefrf} = 9$, with a fixed efficiency of 0.8 Amps per Watt of power injected into the plasma (the coding for this model is, therefore, trivial). The wall plug to injector efficiency is set using input parameter $\text{etaof}$, which has a default value of 0.5, and the unit cost is set using input parameter $\text{ucof}$. The default value for $\text{ucof}$ is 3.3 $ per Watt of injected power.

The bootstrap fraction is assumed to be zero.

Plasma ignition, and additional plasma heating using auxiliary power, are treated as for tokamaks.

3.6.2 TF coils

The TF coils for the RFP option are derived from the TITAN-II coil set, which uses circular copper coils. The radial thickness is set using $\text{tfcth}$ as usual, but the toroidal thickness may also be set, using iteration variable no. 77 ($\text{tftort}$). This is constrained to be no larger than is geometrically possible using constraint equation no. 47 with iteration variable no. 76 ($\text{frfptf}$).
3.6.3 Ohmic heating coils

The TITAN-I ohmic heating coil locations, currents etc. are assumed. These comprise 14 copper coils, with currents swinging from positive to negative during the plasma start-up period, and then decaying back to zero. To account for different plasma geometries and currents, the ohmic heating coil locations relative to the plasma centre scale with the TF coil radius and with the plasma major radius, and the current per turn scales linearly with the plasma current.

3.6.4 EF coils

The Equilibrium Field coils are based on the TITAN-I EF coils, which are two superconducting (NbTi) coils that provide the correct vertical field at the plasma centre. The coil locations scale with the plasma major radius and TF coil radius along with the ohmic heating coils.

3.6.5 Divertor

The TITAN divertor concept uses three poloidal divertors, which bear little resemblance to the typical tokamak toroidal divertor systems. The code uses the TITAN divertor lifetime of one year to enable the divertor costs to be reasonable, although the divertor surface area (and therefore the cost per divertor) is likely to be inaccurate.

3.6.6 Code modifications

As with the stellarator model, the RFP model has been incorporated in such a way as to allow its simple removal again if required in the future. All new routines are confined to the dedicated source file rfp.f90. The tokamak-relevant consistency equations in PROCESS (see Section 2.5) are used without modification, to ensure that the coil currents and the fields they produce are consistent with the plasma parameters.

3.7 Inertial Fusion Energy Model

As well as magnetic confinement devices, PROCESS has the ability to model inertial fusion plants, in which a laser or ion beam is used to ignite a target pellet containing the fusion fuel.

To activate the inertial fusion energy (IFE) coding, it is necessary to create a file device.dat, containing the single character 3 in the first row, in the working directory (see Section 4.4). This has the effect of setting the internally-used switch ife = 1. If the file is absent, or its first character is set to something other than 3, the IFE model is not used, and ife is set to 0.

The IFE model is controlled using two additional switches.

ife typ = 0 : Generic device build
ife typ = 1 : OSIRIS-type device build [55, 56, 57]
ife typ = 2 : SOMBRERO-type device build [58, 59]
ife typ = 3 : HYLIFE-II-type device build [60, 61, 62]
Switch `ifetyp` defines the type of device that is assumed; this varies widely between different conceptual designs. The generic type assumes a cylindrically symmetric device, while the other types are approximations to the builds of the given conceptual machines [63]. In general, the build from the centre of the device (at the target ignition location) is in the order: chamber, first wall, gap, blanket, gap, shield, gap, building wall. The user specifies the thicknesses of these regions, and also the materials that are present and in what proportions.

\(ifedrv = -1\): Driver efficiency and target gain are input as functions of driver energy

\(ifedrv = 0\): Driver efficiency and target gain are input

\(ifedrv = 1\): SOMBRERO laser drive efficiency and target gain assumed

\(ifedrv = 2\): OSIRIS heavy ion beam driver efficiency and target gain are assumed [64]

Switch `ifedrv` defines how the code calculates the driver efficiency and target gain — these are the primary outputs required from the physics part of the model. For the SOMBRERO and OSIRIS cases (\(ifedrv = 1\) and \(ifedrv = 2\), respectively) the driver efficiency and gain are calculated from curves of these parameters as functions of the driver energy. For the \(ifedrv = -1\) case, the user provides the driver efficiency and target gain versus driver energy, via the two arrays `etave(1:10)` and `gainve(1:10)` respectively; the element number corresponds to the driver energy in MJ, and outside the range 1–10 MJ the curves are extrapolated linearly. Finally, for the \(ifedrv = 0\) case, the user inputs single values for the driver efficiency (`drveff`) and target gain (`tgain`).

Constraint equation no. 50 can be turned on to enable the ignition repetition rate to remain below a user-specified upper limit (`rrmax`); iteration variable no. 86 (`frrmax`) is the associated f-value (see Section 2.5). The other iteration variables relevant for the IFE model are nos. 81–85 (`edrive`, `drveff`, `tgain`, `chrad` and `pdrive` — see Table 2.2).

### 3.8 Safety and Environment Models

At present, the neutronics, activation and inventory calculations comprise the safety and environment models in the code.

The models comprising the safety and environmental calculations [65] within the code are all called from routine FISPAC. They are only performed once, at the end of each run, as they take a relatively long time to evaluate, and the results are only used for diagnostic purposes — no constraints are imposed at present to minimise doses, for instance.

N.B. These models are currently not available in the present version of the code.

#### 3.8.1 Neutronics

The neutronics module predicts the neutron flux spectra in the inboard and outboard first wall and blanket components. The spectra are based on a simplified tokamak device that has a fixed ratio (= 1.5825) between the outboard blanket thickness and the inboard blanket thickness, and are scaled according to the actual thickness of the outboard blanket. This relatively limited, single-parameter approach is expected to be replaced by a more general method, which should allow a more accurate portrayal of the device being modelled by PROCESS.
3.8.2 Activation and inventory information

The code evaluates the consequences of exposing the power plant’s materials to the calculated neutron fluxes, subject to the limitations imposed by the neutronics module. A library of neutron cross-sections and decay data is used to calculate the total activity, gamma-ray dose rate and decay heat output due to the materials’ exposure to neutrons, both at the end of the plant’s life and at a time 100 years later. These values are relevant to decommissioning and disposal studies, and additional parameters that can be obtained from the nuclide inventory will also be included as the need arises.

3.9 Cost Models

Two cost models are available, determined by the switch cost_model.

3.9.1 1990 cost model (cost_model = 0)

This combines methods used in the TETRA code and the Generomak scheme. The costs are split into accounting categories. The best references for the algorithms used are [2], and source file costs.f90 in the code itself. The majority of the costed items have a unit cost associated with them. These values scale with (for example) power output, volume, component mass etc., and many are available to be changed via the input file. All costs and their algorithms correspond to 1990 dollars.

The unit costs of the components of the fusion power core are relevant to “first-of-a-kind” items. That is to say, the items are assumed to be relatively expensive to build as they are effectively prototypes and specialised tools and machines have perhaps been made specially to create them. However, if a “production line” has been set up, and R & D progress has allowed more experience to be gained in constructing the power core components, the costs will be reduced as a result. Variable fkind may be used to multiply the raw unit costs of the fusion power core items (by a factor less than one) to simulate this cost reduction for an $N$th-of-a-kind device. In other systems studies of fusion power plants, values for this multiplier have ranged from 0.5 to 0.8.

Many of the unit costs have four possible choices, relating to the level of safety assurance flag lsa. A value lsa = 1 corresponds to a plant with a full safety credit (i.e. is truly passively safe). Levels 2 and 3 lie between the two extremes, and level 4 corresponds to a present day fission reactor, with no safety credit.

The first wall, blanket, divertor, centrepost (if present) and current drive system have relatively short lifetimes because of their hostile environment, after which they must be replaced. Because of this frequent renewal they can be regarded as though they are “fuel” items, and can be costed accordingly. Switch ifueltyp is used to control whether this option is used in the code. If ifueltyp = 1, the costs of the first wall, blanket, divertor and a fraction fcdfuel of the cost of the current drive system are treated as fuel costs. If ifueltyp = 0, these are treated as capital costs.

If the switch ireactor = 0, no cost of electricity calculation is performed. If ireactor = 1, then the cost of electricity is evaluated, with the value quoted in units of $/MWh.

The net electric power is calculated in routine POWER. It is possible that the net electric power can become negative due to a high recirculating power. Switch ipnet determines whether the net electric power is scaled to always remain positive (ipnet = 0), or whether it is allowed to become negative (ipnet = 1), in which case no cost of electricity calculation is performed.
3.9.2 2015 Kovari model (cost_model = 1)

This model [21] provides only capital cost, and is not currently suitable for estimating the cost of electricity. \(N^{th}\)-of-a-kind factors, level of safety assurance factors, and blanket replacement costs are not included. The mean electric output is calculated using the capacity factor, which takes account of the availability and the dwell time for a pulsed reactor. The capital cost divided by the mean electric output is a useful comparison parameter.

3.10 Plant availability

Switch \texttt{iavail} is used to control how the overall plant availability factor \texttt{cfactr} is calculated, as follows

If \texttt{iavail} = 0, the input value of \texttt{cfactr} is used.

If \texttt{iavail} = 1, a model by N. Taylor and D. Ward [71] is used instead, in which \texttt{cfactr} is calculated taking into account the time taken to replace certain components of the fusion power core, and various unplanned unavailability fractions which may be set by the user, as summarised in Table 3.5.

<table>
<thead>
<tr>
<th>input parameter</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{tbktrepl}</td>
<td>time needed to replace blanket (years)</td>
</tr>
<tr>
<td>\texttt{tdivrepl}</td>
<td>time needed to replace divertor (years)</td>
</tr>
<tr>
<td>\texttt{tcomrepl}</td>
<td>time needed to replace both blanket and divertor (years)</td>
</tr>
<tr>
<td>\texttt{uubop}</td>
<td>unplanned unavailability of balance of plant</td>
</tr>
<tr>
<td>\texttt{uucd}</td>
<td>unplanned unavailability of current drive system</td>
</tr>
<tr>
<td>\texttt{uudiv}</td>
<td>unplanned unavailability of divertor</td>
</tr>
<tr>
<td>\texttt{uufuel}</td>
<td>unplanned unavailability of fuelling system</td>
</tr>
<tr>
<td>\texttt{uufw}</td>
<td>unplanned unavailability of first wall</td>
</tr>
<tr>
<td>\texttt{uumag}</td>
<td>unplanned unavailability of magnets</td>
</tr>
<tr>
<td>\texttt{uuves}</td>
<td>unplanned unavailability of vessel</td>
</tr>
</tbody>
</table>

Table 3.5: Summary of the variables in PROCESS that relate to the Taylor-Ward availability model (\texttt{iavail}=1).

If \texttt{iavail} = 2, the new Morris model is implemented [20]. It estimates both planned and unplanned unavailability, and the time during which no power is being generated if the reactor is pulsed.

The planned unavailability is linked to the lifetimes of the blanket and divertor and the time taken to replace them. The lifetime of the blanket is based on the allowable fast neutron fluence. In contrast, the lifetime of the divertor is estimated using the particle and photon load. The time to replace the blanket and divertor have been estimated by Crofts et al, who studied the influence of the number of remote handling systems working in parallel. PROCESS uses a simple fit to their results, and adds a month to allow the dose rate to reduce to an acceptable level before remote handling operations start, and a month to allow for pump-down and preparation for operation at the end of the shutdown.

To estimate the unplanned downtime, each subsystem is represented by a simple model that tries to capture the degradation of reliability when approaching operational and technological limits. This increases the risk of unplanned downtime as design margins are reduced.

For the toroidal field coils, the chance of a quench is likely to be the largest driver of the risk of unplanned unavailability, and this may depend on the temperature margin - the difference between the
actual temperature and the critical temperature of the superconductor. The user inputs a minimum
temperature margin, below which quench is immediate, and a second, higher value, above which
quenches never occur. In between, there is a finite quench rate.

The unplanned downtime for the blanket is based on the number of cycles it experiences before planned
replacement. (This model is restricted to pulsed reactors.) The cycle life of the blanket is expressed
using a reference lifetime. Before this lifetime, there is a constant but small probability of failure in
each pulse. After the reference lifetime the reliability of the blanket starts to decline, reaching zero at
the upper lifetime limit.

It is assumed that the vacuum system can be maintained in parallel with blanket replacement, so it
does not contribute to the planned downtime. The unplanned downtime is based on an assumed failure
rate for a cryopump, and a specified total number of pumps, with some of them being redundant. The
resulting downtime can be reduced to a negligible level if there are several redundant pumps, but in
addition, there is a fixed unavailability to allow for common mode failures affecting several pumps.
Chapter 4

Running PROCESS

The intention of this chapter is to provide a comprehensive prescription for setting up and performing runs with the code. Firstly, the input file’s structure and format is described. The user is then taken through the procedure for setting up the code to model a new machine, and finally an attempt is made to indicate and solve the problems that the user will face whilst trying to achieve a feasible solution.

4.1 Environment set-up

PROCESS is available for execution from the CCFE Fusion Unix Network, on which you must have an account.

To set up your environment to be able to run the code, the user interface and the associated utilities (see Chapter 6), add the following lines to your .bashrc file. This is a file in the user’s home directory, assuming the bash Unix shell is being used. Although hidden, it can be opened by issuing the relevant command, for example gedit .bashrc.

```
module use /home/PROCESS/modules
module swap python
module load process/master
```

(If you want to use the latest draft version of the code instead of the latest verified release version, replace module load process/master with module load process/develop above.)

4.2 User Interface

The user interface allows for viewing and editing of IN.DAT files in a web browser. Launch the interface (GUI) by typing start_gui from the Unix command line.

(The interface uses Django for its web framework. Currently it can only be launched by running the Django development server locally. This is performed automatically if you launch the GUI as above. For instructions on starting the server manually, see the README file in the utilities/processgui directory.)

Please note: If you are working on a Linux desktop machine and already have Firefox open, typing start_gui on a Fusion Unix Network machine will try to open a PROCESS GUI session on the already-open Firefox running locally. This will fail! You need to either close the local Firefox session completely
Chapter 4  Running PROCESS

before typing start_gui or open a different browser manually (e.g. konqueror) after typing start_gui and navigate to 127.0.0.1:8000 in the address bar.

Every variable is listed along with its current value, a 'reference' value and a short description. Hovering over the short description will provide a longer description. Until an input file is opened, all values are given their default values assigned by PROCESS.

The reference values are used to compare differences between two files. Differences between the reference value and the current input value will be highlighted in red.

The 'Meta' section allows editing of the run description, which will be placed at the top of the created IN.DAT. Iteration variables and constraint equations can be enabled using the checkboxes in the relevant sections.

An input file and a reference file can be opened using the buttons at the top of the window. The files opened must be compatible with the PROCESS version number shown in the top right of the window.

The 'Save' button creates an IN.DAT file with a similar layout to the GUI. Variables whose values are set different from the PROCESS default are listed under module headings, along with a comment describing the variable. For integer variables, a description of the value selected may also be included. The variables neqns (number of constraints) and nvar (number of iteration variables) are automatically calculated.

When using the Firefox web browser it can save time to select
Preferences > General > Downloads > Always ask me where to save files.

You can searching for a particular variable can be done using the browser’s in-built search. Use the "Expand" button in the top-right of the window to expand every module heading and use Ctrl-f to search.

4.3 Executing the Code

Execute PROCESS by simply typing `process` on the Unix command line.

By default, the input and output file names are as described in the following sections. If, however, PROCESS is executed with an argument, this is used as a prefix to the file names:

```
process myrun
```

will assume that the input file is called `myrun.IN.DAT`, etc.

The input file must be in the current directory. The output files `myrunOUT.DAT` and `myrunMFILE.DAT` will be created in the current directory.

4.4 The Input File

The input file `IN.DAT` is used to change the values of the physics, engineering and other code parameters from their default values, and to set up the numerics (constraint equations, iteration variables etc.) required to define the problem to be solved. The user interface writes the input file, so it is not necessary to edit it directly. Details of layout and format are in Appendix B.

If the code encounters a problem reading the input file, it will stop immediately with an error message. The last line of the output file `OUT.DAT` may give an indication of where in the input file the problem lies.

4.5 The Output Files

The main output from the code is sent to file `OUT.DAT` in the working directory. It is essential to check that the code reports that it has found a feasible solution.

A second file, `MFILE.DAT`, is also produced in the working directory. This file contains most of the same data as `OUT.DAT` but in a different format, and has been designed to be “machine-readable” by some of the utility programs described in Chapter [6] to allow simple post-processing and graphical output to be produced easily.

4.6 Optimisation mode

Switch `ioptimz` should be set to 1 for optimisation mode.

If `ioptimz = 0`, a non-optimisation pass is performed first. Occasionally this provides a feasible set of initial conditions that aids convergence of the optimiser, but it is recommended to use `ioptimz = 1`. 
Enable all the relevant consistency equations, and it is advisable to enable the corresponding iteration variables shown in bold in Table 2.3. A number of limit equations (inequality constraints) can also be activated. For limit equations, the corresponding f-value must be selected as an iteration variable. In optimisation mode, the number of iteration variables is unlimited.

It may still be difficult, if not impossible, to reconcile the fusion power and the net electric power with the required values. This may well be due to the power conversion efficiency values being used — refer to Figure 3.9.

With luck, a few iterations of this process will produce an adequate benchmark case. A typical input file for use with PROCESS in non-optimisation mode is contained in Appendix D.

If scans of a given variable are to be made over a large range of values, it is often a good idea to start the scan in the middle of the desired range, and to split the scan in two — one going downwards from the initial value, and the other upwards. This ensures that the whole range of the scan produces well-converged machines (assuming a “good” initial point), without sharp changes in gradient in the parameter values.

It should be remembered that the value of the scan variable is set in the array sweep, and this overrules any value set for the variable elsewhere in the input file.

The output from an optimisation run contains an indication as to which iteration variables lie at their limiting values. A typical input file for use with PROCESS in optimisation mode is contained in Appendix C.

4.7 Non-optimisation mode

Non-optimisation mode is sometimes used to perform benchmark comparisons, whereby the machine size, output power etc. are known and one only wishes to find the calculated stresses, beta values and fusion powers, for example.

Running PROCESS in non-optimisation mode requires few changes to be made to the input file from the non-optimisation case. The main differences between optimisation mode and non-optimisation mode are:

1. Non-optimisation mode does NOT apply lower or upper bounds to the iteration variables. It follows that limit equations are not enforced.

2. In non-optimisation mode the number of active iteration variables must be equal to the number of constraints.

3. A figure of merit is not available in non-optimisation mode.

4. Scans cannot be performed in non-optimisation mode.

As before, the user must decide which constraint equations and iteration variables to activate.

4.8 Troubleshooting

Experience has shown that the first few attempts at running PROCESS with a new input file tends to produce unfeasible results — that is, the code will not find a consistent set of machine parameters. The highly non-linear nature of the numerics of PROCESS is the reason for this difficulty, and it often
requires a great deal of painstaking adjustment of the input file to overcome. The utility a_to_b (subsection 6.1.12) is useful in this situation.

### 4.8.1 Error handling

In general, errors detected during a run are handled in a consistent manner, with the code producing (hopefully) useful diagnostic messages to help the user understand what has happened.

There are three levels of errors and warning that may occur:

- **Level 1**: An *informational* message is produced under certain conditions, for example if the code has modified the user’s input choice for some reason.

- **Level 2**: A *warning* message is produced if a non-fatal situation has occurred that may result in an output case that is inaccurate or unreliable in some way.

- **Level 3**: An *error* message will occur if a severe or fatal error has occurred and the program cannot continue.

These messages are printed on the screen during the course of a run, and those still active at the final (feasible or unfeasible) solution point are also written to the end of the output file (messages encountered during the iteration process are not copied to the output file, as the convergence to a valid solution might resolve some of the warnings produced earlier in the solution process).

The `error_status` variable returns the highest severity level that has been encountered (or zero if no abnormal conditions have been found); if a severe error (level 3) is flagged at any point the program is terminated immediately. The final message number encountered during a run is returned via output variable `error_id`. In addition, with certain messages, a number of diagnostic values may also be given; these can be used to provide extra diagnostic information if the source code is available.

### 4.8.2 General problems

A code of the size and complexity of PROCESS contains myriads of equations and variables. Virtually everything depends indirectly on everything else because of the nature of the code structure, so perhaps it is not surprising that it is often difficult to achieve a successful outcome.

Naturally, problems will occur if some of the parameters become unphysical. For example, if the aspect ratio becomes less than or equal to one, then we must expect problems to appear. For this reason, the bounds on the iteration variables should be selected with care.

Occasionally arithmetic (“NaN”) errors are reported. They usually occur when the code is exploring unphysical values of the parameters, and often suggest that no feasible solution exists for the input file used.

The error messages produced by the code attempt to provide diagnostic information, telling the user where the problem occurs, and also suggest a possible solution. These messages are out of necessity brief, and so cannot promise to lead to a more successful outcome.

There is the option to turn on extra debugging output; to do this, set `verbose = 1` in the input file.

### 4.8.3 Optimisation problems

On reflection it is perhaps surprising that PROCESS ever does manage to find the global minimum figure of merit value, since if there are `nvar` iteration variables active the search is over `nvar-
Chapter 4 Running PROCESS

74

dimensional parameter space, in which there may be many shallow minima of approximately equal depth. Remember that \( nvar \) is usually of the order of twenty.

The machine found by PROCESS may not, therefore, be the absolutely optimal device. It is quite easy to have two or more solutions, with results only a few per cent different, but a long way apart in parameter space. The technique of “stationary” scans is sometimes used in this situation: a scan is requested, but the same value of the scan variable is listed repeatedly.

Scans should be started in the middle of a range of values, to try to keep the scan within the same family of machines. The optimum machine found may otherwise suddenly jump to a new region of parameter space, causing the output variables to seem to vary unpredictably with the scanning variable.

It should be noted that in general the machine produced by PROCESS will always sit against one or more operation limits. If, during a scan, the limit being leant upon changes (i.e. if the machine jumps from leaning on the beta limit to leaning on the density limit) the output parameters may well become discontinuous in gradient, and trends may suddenly change direction.

4.8.4 Unfeasible results

In the numerics section of the output file, the code indicates whether the run produced a feasible or unfeasible result.

The former implies a successful outcome, although it is always worth checking that the sum of the squares of the constraint residuals (\( \text{sqsumsq} \)) is small (\( \sim 10^{-3} \) or less): the code will issue a warning if the solver reports convergence but the value of \( \text{sqsumsq} \) exceeds \( 10^{-2} \). If this occurs, reducing the value of the HYBRD tolerance \( \text{ftol} \) or VMCON tolerance \( \text{epsvmc} \) as appropriate should indicate whether the result is valid or not; the output can usually be trusted if (1) the constraint residues\(^1\) fall as the tolerance is reduced to about \( 10^{-8} \), and (2) the code indicates that a feasible solution is still found.

An unfeasible result occurs if PROCESS cannot find a set of values for the iteration variables which satisfies all the given constraints. In this case, the values of the constraint residues shown in the output give some indication of which constraint equations are not being satisfied — those with the highest residues should be examined further. In optimisation mode, the code also indicates which iteration variables lie at the edge of their allowed range.

Unfeasible runs can be caused by specifying physically incompatible input parameters, using insufficient iteration variables, or by starting the problem with unsuitable values of the iteration variables.

The utility run_process (Section 6.1.2) carries out many runs, changing the starting values of the iteration variables randomly. It stops once a feasible solution is found.

Another approach is to start with an input file that gives a feasible solution, and modify it step by step towards the parameters desired. This process is automated by the utility a_to_b (Section 6.1.12). It is important to choose the right number of useful iteration variables for the problem to be solved — it is possible to activate too many iteration variables as well as too few, some of which may be redundant.

Both optimisation and non-optimisation runs can fail with an error message suggesting that the iteration process is not making good progress. This is likely to be due to the code finding itself unable to escape a region of the parameter space where the minimum in the residuals is significantly above

\(^1\)The constraint residues are the final values of \( c_i \) in the constraint equations — see Section 2.5. The value \( \text{sqsumsq} \) is the square root of the sum of the squares of these residuals.
zero. In this situation, there is either no solution possible (the residuals can therefore never approach zero), or the topology of the local minimum makes it difficult for the code to escape to the global minimum. Again, a helpful technique is to either change the list of iteration variables in use, or to simply modify their initial values to try to help the code avoid such regions.

A technique that occasionally removes problems due to unfeasible results, particularly if an error code ifail = 3 is encountered during an optimisation run, is to adjust slightly one of the limits imposed on the iteration variables, even if the limit in question has not been reached. This subtly alters the gradients computed by the code during the iteration process, and may tip the balance so that the code decides that the device produced is feasible after all. For instance, a certain component’s temperature might be 400 K, and its maximum allowable temperature is 1000 K. Adjusting this limit to 900 K (which will make no difference to the actual temperature) may be enough to persuade the code that it has found a feasible solution.

Similarly, the order in which the constraint equations and iteration variables are stored in the icc and ixc arrays can make the difference between a feasible and unfeasible result. This seemingly illogical behaviour is, sadly, typical of the way in which the code works.

Another technique in such situations may be to change the finite difference step length epsfcn, as this might subtly change the path taken in the approach towards a solution.

It may be the case that the act of satisfying all the required constraints is impossible. No machine can exist if the allowed operating regime is too restrictive, or if two constraint equations require conflicting (non-overlapping) parameter spaces. In this case some relaxation of the requirements is needed for the code to produce a successful machine design.

4.8.5 Hints

The above sections should indicate that it is the complex interplay between the constraint equations and the iteration variables that determines whether the code will be successful at producing a useful result. It can be a somewhat laborious process to arrive at a working case, and (unfortunately, perhaps) experience is often of great value in this situation.

It should be remembered that sufficient iteration variables should be used to solve each constraint equation. For instance, a particular limit equation may be $A \leq B$, i.e. $A = fB$, where the f-value $f$ must lie between zero and one for the relation to be satisfied. However, if none of the iteration variables have any effect on the values of $A$ and $B$, and $A$ happens to be greater than $B$, then PROCESS will clearly not be able to solve the constraint.

The lower and upper bounds of the iteration variables are all available to be changed in the input file. Constraints can be relaxed in a controlled manner by moving these bounds, although in some cases care should be taken to ensure that unphysical values cannot occur. The code indicates which iteration variables lie at the edge of their range.

It is suggested that constraint equations should be added one at a time, with sufficient new iteration variables activated at each step. If the situation becomes unfeasible it can be helpful to reset the initial iteration variable values to those shown in the output from a previous feasible case, and rerun the code.
Chapter 5

Changing the Source Code: New Models, Variables and Constraints

It is often useful to add extra features to the code in order to model new situations. This chapter provides instructions on how to do this, with specific details on how to add various numerics related items to PROCESS.

Please remember to modify the relevant sections and table(s) in this User Guide if changes are made to the source code!

5.1 Source Code Modification

Described here are the general rules that apply when the Fortran source code is modified. See Section 7.3 for instructions on how to commit changes to the PROCESS Git repository and produce new releases. The variable descriptor file is generated from specially-formatted comment lines within the source code (see Section 7.2 for more details). Therefore, it is exceedingly important to keep these lines relevant and in sync with the variables they describe.

5.1.1 Changing the Fortran code

Please ensure that the following rules are adhered to when modifying the PROCESS source code:

1. Keep the layout consistent in with the existing code, including indentation of clauses (if-statements, do-loops, etc.).
2. Use the standard routine header (see below).
3. Always use implicit none and declare all local variables explicitly.
4. Declare all ‘real’ (i.e. floating-point) variables as real(kind(1.0D0)).
5. Ensure all routine arguments have the appropriate attribute intent(in), intent(out) or intent(inout), as necessary.
6. Always write explicit real constants using the scientific D notation, e.g. 1.0D0, 2.3D0, -1.23D6. That is to say, use 1.0D0 and not 1.0 or 1. or 1 when the expression should be using floating-point arithmetic.
A Fortran 90/95 manual, complete with guidelines for good Fortran 90/95 practice, may be found at the following webpage:

http://fusweb1.fusion.ccfe.ac.uk/~pknight/f95notebook.html

5.1.2 Source code documentation

It is critically important to keep the documentation in the source code itself up-to-date, relevant and tidy. Please keep to the following guidelines whenever the source code is modified.

1. Use comments copiously in the code.

2. Use the standard header layout, and do not omit any of the sections. Here is an example subprogram header:

```
! +ad_name culblm
! +ad_summ Beta scaling limit
! +ad_type Subroutine
! +ad_auth P J Knight, CCFE, Culham Science Centre
! +ad_cont N/A
! +ad_args bt : input real : toroidal B-field on plasma axis (T)
! +ad_args dnbeta : input real : Troyon-like g coefficient
! +ad_args plascur : input real : plasma current (A)
! +ad_args rminor : input real : plasma minor axis (m)
! +ad_args betalim : output real : beta limit as defined below
! +ad_desc This subroutine calculates the beta limit, using
! +ad_desc the algorithm documented in AEA FUS 172.
! +ad_desc <P>The limit applies to beta defined with respect to the total B-field.
! +ad_desc Switch ICULBL determines which components of beta to include (see
! +ad_desc routine <A HREF="constraints.html">constraints</A> for coding):
! +ad_desc <UL>
! +ad_desc <P><LI>If ICULBL = 0, then the limit is applied to the total beta
! +ad_desc <P><LI>If ICULBL = 1, then the limit is applied to the thermal beta only
! +ad_desc <P><LI>If ICULBL = 2, then the limit is applied to the thermal +
! +ad_desc neutral beam beta components
! +ad_desc </UL>
! +ad_desc The default value for the g coefficient is DNBETA = 3.5
! +ad_prob None
! +ad_call None
! +ad_hist 21/06/94 PJK Upgrade to higher standard of coding
! +ad_hist 09/11/11 PJK Initial F90 version
! +ad_hist 27/06/13 PJK Modified header comments
! +ad_stat Okay
! +ad_docs AEA FUS 172: Physics Assessment for the European Reactor Study
! +ad_docs AEA FUS 251: A User’s Guide to the PROCESS Systems Code

! +ad_name culblm (bt,dnbeta,plascur,rminor,betalim)
```

A description of all the available automatic documentation marker tags (these all start with !+ad_) may be found by examining the main program header of the (self-documenting!) automatic documentation program itself (in file autodoc.f90).

3. After every change within a routine, add a new history line (using !+ad_hist, and !+ad_hisc for continuation lines) to its header. Ensure that all routines called are listed via !+ad_call
lines. Update the description lines as necessary. You may use html tags and hyperlinks (some are shown in the example above) as required; to be sure that they have been added correctly, type `make html` to create the web documentation and examine the relevant html file (i.e. `culblm.html` for the example above) using your favourite web browser.

4. If you add a new routine to a module, remember to modify the header of the module as well as that of the new routine (add a `!+ad_cont` line to it).

5. Add suitable documentation to this User Guide whenever a model is added or modified. This should be done immediately to ensure that the Guide remains consistent with the source code. Change the code’s revision number and the date in `process.tex`.

6. Add a new file to the folder `release_notes`. It is particularly important to describe here any changes to the required IN.DAT - especially any change that makes previous IN.DAT files unusable.

### 5.2 Input Parameters

Input parameters (see Section [2.3]) are added to the code in the following way:

1. Choose the most relevant module (usually one of those in source file `global_variables.f90`). Keeping everything in alphabetical order (or possibly within a group of variables closely-related to a particular switch), add a declaration statement for the new variable, specifying a “sensible” default value, and a correctly formatted comment line to describe the variable. Copy the examples already present, such as

```fortran
!+ad_vars abktflnc /5.0/ : allowable first wall/blanket neutron
!+ad_varc fluence (MW-yr/m2) (blktmodel=0)
real(kind(1.0D0)) :: abktflnc = 5.0D0
```

Note that the automatic documentation marker tag `!+ad_vars` tells the `autodoc` utility (Section [7.2]) that the line is (the first line of) a variable description, while `!+ad_varc` specifies any continuation lines. Also note that the colon (`:`) on the first line is necessary, as it is assumed to exist by the dictionary-building Python utility for the GUI.

2. Add a dated comment (using `!+ad_hist`) to the header of the relevant module.

3. Ensure that all the modules that use the new variable reference the relevant module via the Fortran `use` statement.

4. Add the parameter to routine `PARSE_INPUT_FILE` in source file `input.f90` in a suitable place — keep to alphabetical order. The existing examples provide guidance on how to do this. Note that real (i.e. double precision) and integer variables are treated differently, as are scalar quantities and arrays.

### 5.3 Iteration Variables

New iteration variables (see Section [2.4]) are added in the same way as input parameters, with the following additions:
1. Increment the parameter `ipnvars` in module `numerics` in source file `numerics.f90` in order to accommodate the new iteration variable.

2. Add an additional line to the initialisation of the array `ixc` in module `numerics` in source file `numerics.f90`.

3. Assign sensible values for the variable’s bounds to the relevant elements in arrays `boundl` and `boundu` in module `numerics` in source file `numerics.f90`.

4. Assign the relevant element of character array `lablxc` to the name of the variable, in module `numerics` in source file `numerics.f90`.

5. Add the variable to routines `LOADXC` and `CONVXC` in source file `iteration_variables.f90`, mimicking the way that the existing iteration variables are coded.

6. Modify the variable’s description in its declaring module (which is likely to be in source file `global_variables.f90`), to say that it is “(iteration variable XXX)”

7. Add dated comments (using `!+ad_hist`) to the headers of all affected routines and modules.

8. Modify Table 2.2 of this User Guide (`UserGuide_Chapter2.tex`).

If an existing input parameter is now required to be an iteration variable, then simply carry out the tasks mentioned here.

It should be noted that iteration variables must not be reset elsewhere in the code. That is, they may only be assigned new values when originally initialised (in the relevant module, or in the input file if required), and in routine `CONVXC` where the iteration process itself is performed. Otherwise, the numerical procedure cannot adjust the value as it requires, and the program will fail.

### 5.4 Other Global Variables

This type of variable embraces all those present in the modules in `global_variables.f90` (and some others elsewhere) which do not need to be given initial values or to be input, as they are calculated within the code. These should be added to the code in the following way:

1. Choose the most relevant module (usually one of those in source file `global_variables.f90`).
   Keeping everything in alphabetical order (or possibly within a group of variables closely-related to a particular switch), add a declaration statement for the new variable, specifying the initial value `0.0D0`, and a correctly formatted comment line to describe the variable (copying the examples already present — see also “Input Parameters” above).

2. Ensure that all the modules that use the new variable reference the relevant module via the Fortran `use` statement.

### 5.5 Constraint Equations

Constraint equations (see Section 2.5) are added to `PROCESS` in the following way:

1. Increment the parameter `ipeqns` in module `numerics` in source file `numerics.f90` in order to accommodate the new constraint.
2. Add an additional line to the initialisation of the array \texttt{icc} in module \texttt{numerics} in source file \texttt{numerics.f90}.

3. Assign a description of the new constraint to the relevant element of array \texttt{lablcc}, in module \texttt{numerics} in source file \texttt{numerics.f90}.

4. Add a new Fortran \texttt{case} statement containing the new constraint equation to routine \texttt{CONSTRAINT_EQNS} in source file \texttt{constraint_equations.f90}, ensuring that all the variables used in the formula are contained in the modules specified via \texttt{use} statements present at the start of this file. Use a similar formulation to that used for the existing constraint equations, remembering that the code will try to force \textit{cc}(i) to be zero.

5. Add dated comments (using \texttt{!+ad_hist}) to the headers of all affected routines and modules.

6. Modify Table 2.4 of this User Guide (\texttt{UserGuide\_Chapter2.tex}).

Remember that if a limit equation is being added, a new \textit{f}-value iteration variable may also need to be added to the code.

### 5.6 Figures of Merit

New figures of merit (see Section 2.6) are added to \texttt{PROCESS} in the following way:

1. Increment the parameter \texttt{ipnfoms} in module \texttt{numerics} in source file \texttt{numerics.f90} to accommodate the new figure of merit.

2. Assign a description of the new figure of merit to the relevant element of array \texttt{lablmm} in module \texttt{numerics} in source file \texttt{numerics.f90}.

3. Add the new figure of merit equation to routine \texttt{FUNFOM} in source file \texttt{evaluators.f90}, following the method used in the existing examples. The value of \textit{fc} should be of order unity, so select a reasonable scaling factor if necessary. Ensure that all the variables used in the new equation are contained in the modules specified via \texttt{use} statements present at the start of this file.

4. Add dated comments (using \texttt{!+ad_hist}) to the headers of all affected routines and modules.

5. Modify Table 2.5 of this User Guide (\texttt{UserGuide\_Chapter2.tex}).

### 5.7 Scanning Variables

Scanning variables (see Section 2.7) are added to \texttt{PROCESS} in the following way:

1. Increment the parameter \texttt{ipnscnv} in module \texttt{scan\_module} in source file \texttt{scan.f90} to accommodate the new scanning variable.

2. Add a short description of the new scanning variable to the \texttt{nsweep} entry in source file \texttt{scan.f90}.

3. Add a new assignment to the relevant part of routine \texttt{SCAN} in source file \texttt{scan.f90}, following the examples already present, including the inclusion of a short description of the new scanning variable in variable \texttt{xlabel}. 

4. Ensure that the scanning variable used in the assignment is contained in one of the modules specified via `use` statements present at the start of this routine.

5. Add dated comments (using `!+ad_hist`) to the headers of all affected routines and modules.

6. Modify Table 2.6 of this User Guide (UserGuide_Chapter2.tex).

### 5.8 Submission of New Models

The `PROCESS` source code is maintained by CCFE, and resides in a `Git` repository on the CCFE servers. We welcome contributions of alternative or improved models and algorithms.

We request that contributors provide the following information for any new models that they provide:

- A comprehensive description of the model; please provide a full list of references.
- A list of all inputs and outputs: descriptions, default (input) values, allowed ranges, units.
- If possible, please cross-reference any input/output variables to existing global variables listed in the variable descriptor file (see Section 2.2).
- Any new input parameters, iteration variables, constraint equations, figures of merit etc.
- A definition of any pre-requisites.
- Any available test data, code examples or test programs in any language.

### 5.9 Code Structure

#### 5.9.1 Numerics modules

These modules contain the equation solvers, their calling routines and other relevant procedures. Various mathematical routines from a number of standard libraries are also incorporated into these files. Table 5.1 summarises the numerics source files.

<table>
<thead>
<tr>
<th>source file</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>caller.f90</td>
<td>calls physics, engineering, building and cost routines</td>
</tr>
<tr>
<td>constraint_equations.f90</td>
<td>defines the constraint equations</td>
</tr>
<tr>
<td>evaluators.f90</td>
<td>function evaluators for HYBRD and VMCON packages</td>
</tr>
<tr>
<td>iteration_variables.f90</td>
<td>adjusts values of iteration variables</td>
</tr>
<tr>
<td>maths_library.f90</td>
<td>miscellaneous ‘black-box’ maths routines, including HYBRD and VMCON packages</td>
</tr>
<tr>
<td>numerics.f90</td>
<td>numerics array definitions, and calling routines for HYBRD and VMCON packages</td>
</tr>
<tr>
<td>scan.f90</td>
<td>performs a parameter scan</td>
</tr>
</tbody>
</table>

Table 5.1: *Summary of the numerics modules in PROCESS.*

#### 5.9.2 Physics modules

These modules contain the main physics routines that evaluate the plasma and fusion parameters. Also included here are the routines describing the current drive and divertor systems. Table 5.2 summarises the main physics source files.
### Table 5.2: Summary of the physics modules in PROCESS.

<table>
<thead>
<tr>
<th>source file</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>current_drive.f90</td>
<td>current drive efficiency calculations</td>
</tr>
<tr>
<td>divertor.f90</td>
<td>divertor model calculations</td>
</tr>
<tr>
<td>fispact.f90</td>
<td>nuclide inventory/activation calculations</td>
</tr>
<tr>
<td>ife.f90</td>
<td>inertial fusion energy physics/engineering</td>
</tr>
<tr>
<td>impurity_radiation.f90</td>
<td>radiation power calculations</td>
</tr>
<tr>
<td>physics.f90</td>
<td>tokamak plasma and fusion calculations</td>
</tr>
<tr>
<td>plasma_geometry.f90</td>
<td>plasma geometry algorithms</td>
</tr>
<tr>
<td>plasma_profiles.f90</td>
<td>plasma density and temperature profile calculations</td>
</tr>
<tr>
<td>rfp.f90</td>
<td>reversed field pinch physics/engineering</td>
</tr>
<tr>
<td>startup.f90</td>
<td>plasma start-up auxiliary power requirements</td>
</tr>
<tr>
<td>stellarator.f90</td>
<td>stellarator-relevant physics/engineering</td>
</tr>
</tbody>
</table>

### 5.9.3 Engineering modules

These modules contain the description of the machine geometry and its major systems, including the PF and TF coil sets, the first wall, blanket and shield, and other items such as the buildings, vacuum system, power conversion and the structural components. Table 5.3 summarises the main engineering source files.

<table>
<thead>
<tr>
<th>source file</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>availability.f90</td>
<td>plant component lifetimes and overall availability</td>
</tr>
<tr>
<td>buildings.f90</td>
<td>buildings calculations</td>
</tr>
<tr>
<td>fwbs.f90</td>
<td>first wall, blanket and shield calculations</td>
</tr>
<tr>
<td>machine_build.f90</td>
<td>machine build calculations</td>
</tr>
<tr>
<td>pfcoil.f90</td>
<td>PF coil calculations</td>
</tr>
<tr>
<td>plant_power.f90</td>
<td>heat transport and power balance calculations</td>
</tr>
<tr>
<td>pulse.f90</td>
<td>pulsed power plant calculations</td>
</tr>
<tr>
<td>safety.f90</td>
<td>steady-state temperatures after a LOCA event</td>
</tr>
<tr>
<td>sctfcoil.f90</td>
<td>superconducting TF coil calculations</td>
</tr>
<tr>
<td>structure.f90</td>
<td>support structure calculations</td>
</tr>
<tr>
<td>tfoil.f90</td>
<td>resistive TF coil calculations</td>
</tr>
<tr>
<td>vacuum.f90</td>
<td>vacuum system calculations</td>
</tr>
</tbody>
</table>

### Table 5.3: Summary of the engineering modules in PROCESS.

### 5.9.4 Costing module

The costing module, costs.f90, performs all the cost calculations, including values in M$ for each machine system, and the cost of electricity in m$/kWh. Normally, the machine costs are written to the output file; if this is not required set switch `output_costs = 0`.

### 5.9.5 Other modules

These modules perform miscellaneous tasks, such as initialisation of variables and file input / output. File process.f90 contains the main program, and includes the overall controlling loop.
Table 5.4 summarises these modules.

<table>
<thead>
<tr>
<th>source file</th>
<th>description</th>
</tr>
</thead>
<tbody>
<tr>
<td>error_handling.f90</td>
<td>centralised error handling module</td>
</tr>
<tr>
<td>fson_library.f90</td>
<td>library used to read in data from JSON-format files</td>
</tr>
<tr>
<td>global_variables.f90</td>
<td>defines and initialises most shared variables</td>
</tr>
<tr>
<td>initial.f90</td>
<td>checks self-consistency of input variables and switches</td>
</tr>
<tr>
<td>input.f90</td>
<td>reads in user-defined settings from input file</td>
</tr>
<tr>
<td>output.f90</td>
<td>utility routines to format output to file</td>
</tr>
<tr>
<td>process.f90</td>
<td>main program and top-level calling routines</td>
</tr>
</tbody>
</table>

Table 5.4: *Summary of the remaining modules in PROCESS.*
Chapter 6

Utility Programs

A number of utilities for PROCESS are available, for instance to modify the input file IN.DAT, run PROCESS until a feasible solution is found, or to extract and plot data from the PROCESS output. All utilities are available from PROCESS/master/utilities.

For PROCESS users, only the executables described in Section 6.1 are relevant. For anyone interested in modifying the existing code or developing new utilities the PROCESS Python libraries are described in Section 6.2.

All executables use Python library functions either from the publicly available numpy, scipy and matplotlib libraries or the PROCESS Python libraries documented in Section 6.2. To use the PROCESS Python libraries you need to make sure their directory is in your Python path. Use the commands given in Section 4.1 to do this.

All Python code has been written for Python 3.

6.1 Executables

All executables can be run by using any of the following commands:

```
python executable_name.py
python3 executable_name.py
executable_name.py
```

and they typically have a short description of their usage when putting -h or --help as arguments:

```
executable_name.py -h
```

Some executables come with a configuration file that typically follows the naming convention executable_name.conf. To run the executable please copy the config file into your own work directory. Do NOT try to edit the config file in the PROCESS directory.

6.1.1 Turn output into input (write_new_in_dat.py)

This program modifies a given IN.DAT file such that the specified initial values of all the iteration variables are given by their values in OUT.DAT.
Input: IN.DAT, OUT.DAT
Output: new_IN.DAT

6.1.2 Randomly vary the starting point (run_process.py)

This program runs PROCESS many times, randomly varying the initial values of the iteration variables until a feasible solution is found. Stops when a feasible solution is found. If in the course of a scan some scan points result in feasible solutions and some do not, run_process.py will stop if and only if the number of unfeasible scan points is less than or equal to the parameter NO_ALLOWED_UNFEASIBLE.

Input: run_process.conf (optional), IN.DAT
Output: A directory is created for each run. containing all the standard PROCESS output, process.log (log file), and README.txt (contains comments from config file)

Configuration Options: The configuration file run_process.conf has the following style:

* This is a comment in the config file!

* Path to working directory in which PROCESS is run.
  WDIR = Run1
* original IN.DAT name
  ORIGINAL_IN_DAT = demo1a_10_sep_13.IN.DAT
* PATH to PROCESS binary
  PROCESS = ~PROCESS/master/process
* ONE line comment to be put into README.txt
  COMMENT = This is a test DEMO run! ;-) 
* Maximum number of PROCESS runs
  NITER = 5
* integer seed for random number generator; use None for random seed
  SEED = 2
* factor within which the iteration variables are changed
  FACTOR = 1.5
* Number of allowed unfeasible points that do not trigger rerunning.
  NO_ALLOWED_UNFEASIBLE = 0
* include a summary file with the iteration variables at each stage.
  INCLUDE_ITERVAR_DIFF = True
* add iteration variables - comma separated
  ADD_IXC = 99, 77
* remove iteration variables - comma separated
  DEL_IXC =
* add constraint equations - comma separated
  ADD_ICC = 57,
* remove constraint equations - comma separated
  DEL_ICC =
* set any variable to a new value
* the variable does not have to exist in IN.DAT
VAR_TFTORT = 1.9
VAR_EPSVMC = 1e-4
* remove variables
DEL_VAR_IITER

A configuration file with an alternative name can be specified using the optional argument

    run_process.py -f CONFIGFILE

### 6.1.3 List PROCESS runs with comments (build_index.py)

Creates an index of all PROCESS run comments, after they have been created by run_process in a series of subfolders.

**Input:** None

**Output:** Index.txt

**Configuration Options:** Optional arguments are:

```
# change the name of the file containing the folder description
build_index.py -r README.txt
# append the results to Index.txt instead of creating a new file
build_index.py -m a
# change the name of the subfolder Base - default=Run
build_index.py -b Base
# give a list of subfolder suffixes - default=all
build_index.py -s 1-4,6,8,9-12
# increase verbosity
build_index.py -v
```

An example Index.txt file might look like this

```markdown
Run1:
Original run

Run2:
Changed the no. TF coils
...```
6.1.4 test_process.py

This program has a similar structure to the run_process.py program, but varies the iteration variable start parameters for as many iterations as requested by the configuration file. It outputs a summary of the final iteration parameter values and figure of merit values for testing the accuracy of the optimisation solver in PROCESS.

**Input:** test_process.conf, an IN.DAT file as specified in the config file

**Output:** All the standard PROCESS output, process.log, README.txt, time.info (stores the run time of actual PROCESS iterations), SolverTest.out

The SolverTest.out file contains a table of the ifail values (see Table A.1), the figure of merit, the square root of the sum of the squares of the individual constraints as well as the initial and final values for the iteration variables and the individual constraint residuals. The value of Q, i.e. the ratio of fusion power to injected power, is also included in the SolverTest.out file. All values can be used to diagnose the performance of the optimisation solver.

**Configuration Options:** The configuration file test_process.conf has the following style:

```plaintext
* No iterations
NITER = 1000

* Working directory: Subdirectory in which to copy IN.DAT and test_process.conf
WDIR=Run1/

* original IN.DAT name (should not be called IN.DAT!)
ORIGINAL_IN_DAT = minimal_demo2_nosweep.IN.DAT

* sets flag of same name in IN.DAT:
* None - keeps original value
* -1 - Non-optimisation only
* 0 - one non-optimisation step followed by optimised iteration
* 1 - optimisation only
IOPTIMZ = 1

* sets the error tolerance for VMCON
* None - keeps the original value (IN.DAT or default)
EPSVMC = None

* sets the step length for the finite difference derivatives
* None - keeps the original value (IN.DAT or default)
EPSFCN = None

*sets the figure of merit
*positive value - minimise, negative value - maximise
```
* None - keeps the original value
* 1   - plasma major radius
* 6   - cost of electricity
MINMAX = None

* integer seed for random number generator; use None for random seed
SEED = 2

* factor within which the iteration variables are changed
FACTOR = 1.1

* PATH to PROCESS binary
PROCESS=process.exe

A configuration file with an alternative name can be specified using the optional argument

test_process.py -f CONFIGFILE

6.1.5 Output plotting: create data file (make_plot_dat.py)

Creates a PLOT.DAT-type file from MFILE.DAT. This is required by plot_sweep.py.

Input: make_plot_dat.conf, MFILE.DAT
Output: make_plot_dat.out

Configuration Options: Optional arguments are:

    # new variables for output
    make_plot_dat.py -p rmajor
    # writes make_plot_dat.out in columns
    make_plot_dat.py --columns
    # resets make_plot_dat.conf to PLOT.DAT layout
    make_plot_dat.py --reset-config
    # file to read as input
    make_plot_dat.py -f MFILE.DAT
    # run with default parameters
    make_plot_dat.py --defaults

An example version of make_plot_dat.conf might look like this:

    # make_plot_dat.out config file.
    rmajor
    aspect
6.1.6 Create a two-page summary (plot_proc.py)

A utility to produce a two-page summary of the output, including the major parameters, poloidal and toroidal cross-sections, and temperature and density profiles.

**Input:** MFILE.DAT

**Output:** proc_plot_out.pdf (or as specified by user)

**Configuration Options:** Optional arguments are:

```
# change the input file name
python plot_proc.py -f MFILE.DAT
# change the output file name
python plot_proc.py -o out.pdf
# show the plot as well as saving the figure
python plot_proc.py -s
```

6.1.7 output_data.py

A utility to output a set of data very similar to plot_proc.py, but to a comma-delimited format for inclusion in spreadsheets. Not recommended - it's best to use PLOT.DAT instead, as this is always generated by PROCESS, and can be easily loaded into a spreadsheet.

**Input:** MFILE.DAT (or as specified by user)

**Output:** process\_summary.txt (or as specified by user)

**Configuration Options:** Optional arguments are:

```
# change the input file name
python output_data.py -f MFILE.DAT
# change the output file name
python output_data.py -o out.txt
```
6.1.8 plot_sweep.py

This utility plots normalised values from PLOT.DAT and allows comparisons of changes in values from a sweep compared to the first point in the sweep. It is now deprecated due to the existence of more flexible sweep-generating and output-handling utilities.

**Input:** PLOT.DAT as output by make_plot_dat.py

**Output:** PLOT.DAT.eps (default or as specified by user)

**Configuration Options:** Optional arguments are:

```
# creates PLOT.DAT.eps with R0, IP, beta
python plot_sweep.py -g 11 13 18
# creates demo1.png with Te, n
python plot_sweep.py -o demo1.png 21 22
```

6.1.9 Plot scan results (plot_mfile_sweep.py)

This utility plots normalised values of the iteration variables output by a parameter scan. Zero indicates an iteration variable at its lower bound and 1 an iteration variable at its upper bound.

**Input:** MFILE.DAT

**Output:** sweep_fig.pdf (default or as specified by user)

Optional arguments are:

```
# creates sweep_fig.pdf with R0, te, aspect (same variable names as in MFILE.DAT)
python plot_mfile_sweep.py -p rmajor te aspect
# creates demo1.png with Te, n
python plot_mfile_sweep.py -o demo1.png -p te dene
# creates a sweep_fig.pdf with R0, aspect with a different MFILE.DAT
python plot_mfile_sweep.py -f diff_mfile.dat -p rmajor aspect
# Show plot to screen instead of saving with R0 and aspect
python plot_mfile_sweep.py -p rmajor aspect --show
```

Use -h or --help for help

6.1.10 Plot iteration variables and constraint residuals (diagnose_process.py)

This utility aids the user to interpret PROCESS runs that do not find a feasible solution (unless PROCESS has terminated prematurely). It reads the MFILE.DAT and plots the normalised iteration variables, i.e. the iteration variable values normalised to their bounds such that 0 indicates an iteration variable at its lower bound and 1 an iteration variable at its upper bound. Furthermore, it shows the normalised constraint residuals.

**Input:** MFILE.DAT
Output: Displays plots on screen, still need to be saved by the user! (Remember to use -Y or -X, if sshing into a remote machine!)

Optional arguments are:

```
# allows to specify another location/name for the MFILE
python diagnose_process.py -f MFILE.DAT

Use -h or --help for help
```

6.1.11 fit_profile.py

This is a Python tool to fit a general temperature or density profile as given by the pedestalised profile parametrisation (ipedestal =1) to an ascii table. It is using a least squares method and is fitting the position of the pedestal as well as the peaking factors. Optional arguments are

```
-h, --help       show this help message and exit
-f FILENAME, --filename FILENAME
ascii file containing data in columns, default = profile.txt
-r RHO, --rho RHO column of the normalised radius rho=r/a, default = 0
-n DENSITY, --density DENSITY
column of the density profile, default = 1
-t TEMPERATURE, --temperature TEMPERATURE
column of the temperature profile, default = 2
-rn RHOPEDN, --rhopedn RHOPEDN
user defined initial guess of the density pedestal position, if outside [0,1] starts at 0.9, default = 0.9
-rt RHOPEDT, --rhopedt RHOPEDT
user defined initial guess of the temperature pedestal position, if outside [0,1], starts at 0.9, default = 0.9
```

If the column of the density or temperature data does not exist, it is ignored. A warning is issued.

6.1.12 Step from one model to another (a_to_b.py)

When attempting to model a reactor very different from those in previous studies, it can be difficult to find a feasible solution. This utility takes an initial IN.DAT (A.DAT) that is known to run successfully, and a target IN.DAT (B.DAT) that is substantially different. The difference is broken into small steps, and PROCESS runs repeatedly, stepping from the initial input file to the target input file. After each step, the output values of the iteration variables are used as the input starting values for the next step.

Input: a_to_b.conf, A.DAT and its corresponding MFILE.DAT, B.DAT. (These names can be changed by the user.)
Output

When the program finishes, the .DAT files from the last run of `PROCESS` can be found in the specified working directory.

If option `keep_output = True`, then `IN.DAT`, `OUT.DAT`, `MFILE.DAT` and logged `PROCESS` output of each step are stored. Files are prefaced with their step number eg. `003.IN.DAT` and copied to the specified output directory.

Configuration Options: The configuration file `a_to_b.conf` has the following style:

```
*Comment line

*Working directory to store temporary files, default = wdir
wdir = wdir

*Switch to keep .DAT files for every step, default = True
keep_output = True

*Directory for output if keep_output = True, default = steps
outdir = steps

*IN.DAT file for A, default = A.DAT
a_filename = A.DAT

*IN.DAT file for B, default = B.DAT
b_filename = B.DAT

*Path to process binary
path_to_process = /home/PROCESS/master/process

*Number of iterations of vary_iteration_variables to run, default = 20
vary_niter = 20

*Number of steps to go from A to B, default = 10
nsteps = 10

*Factor to vary iteration variables within, default = 1.2
factor = 1.2

*Gap between upper and lower bounds to narrow to, default = 1.001
bound_gap = 1.001
```
6.1.13 N-Dimensional Scanner Utility

This suite of Python utilities allows the user to conduct systematic, multi-dimensional parameter studies with PROCESS. It systematically varies a set of N user defined parameters within predefined bounds. The final results can be evaluated using the corresponding visualisation tool and/or saved in standard NetCDF format for further analysis.

The suite contains the following executables

- ndscan.py - executes the Nd-scan as specified in the configuration file.
- ndscan_package_only.py - creates a NetCDF output file from a previous Nd-scan run.
- ndscan_and_package.py - both executes the Nd-scan and creates the NetCDF output file.
- ndscan_visualisation.py - visualises the NetCDF output.

**Input:** ndscan.json, IN.DAT

**Output:** All MFILES in subdirectory MFILES, packaged NetCDF output file as named in configuration file (default “NdscanOutput.nc”).

When running any of the ndscan tools, optional arguments are:

```bash
#to specify another location/name for the configfile
ndscan.py -f CONFIGFILE

Use -h or --help for help
```

For the visualisation tool the corresponding NetCDF input file can also be specified with `-f`. Per default “NdscanOutput.nc” is used.

**Configuration Options:** The configuration file ndscan.json uses the JSON format\(^1\) and has the following style

```json
{
    "axes": [
    {
        "lowerbound": 7.5,
        "steps": 16,
        "upperbound": 9.0,
        "varname": "rmajor"
    },
    {
        "lowerbound": 5.5,
        "steps": 16,
        "upperbound": 7.0,
```
The only required input parameter in the configuration file are the scan axes, out of which at least one has to be specified. For each axis a `varname`, a `lowerbound`, an `upperbound` and the number of `steps` > 1 have to be specified. All other parameters in the configuration file are optional. The parameters relevant for running the N-dimensional scan are:

```json
{
  "varname": "bt",
  "description": "Description of the goals of this specific run",
  "title": "NdscanOutput",
  "author": "Me",
  "output_vars": [
    "beta",
    "pheat",
    "powfmw",
    "pradmu",
    "powerht",
    "cirpowfr",
    "te",
    "hfact",
    "dnelint",
    "dene",
    "rmajor",
    "bt",
    "pnetelmw",
    "coe",
    "fwbllife",
    "capcost",
    "palmw",
    "wallmu",
    "taueff"
  ],
  "optionals": {
    "remove_scanvars_from_ixc": true,
    "smooth_itervars": true
  }
}
```

 Anything in the comment section (like all other undefined sections) is for the user only and
will be ignored by the program.

**optionals:** `remove_scanvars_from_ixc` Removes all scanning variables from the iteration variables of the IN.DAT file (default=True).

**optionals:** `smooth_itervars` Ensures that each next point starts from the last successful run. This increases the run time, but improves convergence and reduces errors.

The parameters only relevant to the creation of the summary NetCDF file are:

**author** The author will be copied into the NetCDF file.

**description** The description will be copied into the NetCDF file.

**title** Name of the output NetCDF file (default NdscanOutput) that is also copied into the title of the NetCDF file.

**output_vars** The variables that will be extracted from the MFILEs and stored in the NetCDF file. (Only needed when creating the NetCDF output file.)

Additional parameters can be specified as in the config section for the `evaluate_uncertainties.py` tool, see section 6.1.14.1.

The resulting NetCDF file can be visualised using the `ndscan_visualisation.py` tool. It has an interactive menu and is fairly self-explanatory.

### 6.1.14 Uncertainty Tools

In this section, we explain the usage of the PROCESS tools to both evaluate the uncertainties of a design point as well as display them using a simple plotting facility.

Note that the uncertainty evaluation tool has a significantly longer run time than typical evaluations of PROCESS design points and therefore should only be used once a suitable design point has been found. As only user selected output data is kept, the user is recommended to put careful thought into the list of needed output variables.

#### 6.1.14.1 `evaluate_uncertainties.py`

This program evaluates the uncertainties of a single PROCESS design point by use of a Monte Carlo method as described in [83]. It takes a set of uncertain parameters as input, each of which can have either a Gaussian or a flat ('top-hat') probability distribution. These are specified together with optional details about the PROCESS run in a configuration file. Additionally, an IN.DAT file describing the relevant design point needs to be present. (If necessary this can be created from an MFILE.DAT by using the `write_new_indat.py` tool.)

When running the `evaluate_uncertainties.py` tool, optional arguments are:

```bash
# to specify another location/name for the configfile
evaluate_uncertainties.py -f CONFIGFILE

Use -h or --help for help
```
Input: `evaluate_uncertainties.json`,

IN.DAT (or an alternative input file as specified in the config file)

Output:

- uncertainties.nc. This file (NetCDF format) can be visualised using the `display_uncertainties.py` tool.
- Readme.txt,
- process.log,
- PROCESS output from the last run.

The configuration file `evaluate_uncertainties.json` uses the JSON format\(^2\) and has the following style:

```
{
  "_description": "Config file for uncertainties evaluation",
  "_author": "Hanni Lux",

  "config": {
    "runtitle": "testrun for uncertainty tool on DEMO2",
    "IN.DAT_path": "IN.DAT_demo2",
    "process_bin": "PROCESS/master/process.exe",
    "working_directory": "Run1",
    "pseudorandom_seed": 2
  },

  "uncertainties": [
    {
      "Varname": "flhthresh",
      "Errortype": "Gaussian",
      "Mean": 1.0,
      "Std": 0.05
    },
    {
      "Varname": "coreradius",
      "Errortype": "Uniform",
      "Lowerbound": 0.6,
      "Upperbound": 0.9
    },
    {
      "Varname": "etanbi",
      "Errortype": "Relative",
      "Mean": 0.3,
      "Percentage": 10.0
    },
    {
      "Varname": "boundu(9)",
      "Errortype": "LowerHalfGaussian",
    }
  ]
}
```

\(^2\)www.json.org
By convention, we have designated metadata about the PROCESS runs as having a preceding underscore to distinguish these values from the other configuration data used directly by the tools or PROCESS itself. Furthermore, all the optional attributes that can be changed when running PROCESS from most Python utilities like e.g. run_process.py can be specified in the “config” section. All these values have default values and do not need to be set.

runTitle is a one line description of the purpose of the run to be saved in Readme.txt in the working directory as well as the runTitle parameter in the OUT.DAT and MFILE.DAT files. Per default it is empty.

IN.DAT_path is the name/path of the IN.DAT file describing the design point. If not specified it is assumed to be IN.DAT.

process_bin is the process binary that should be used. The default assumes that the user works on the CCFE Fusion Linux machines and has executed the module commands for PROCESS as described in 4.1. Then either the master or development branch of process is being used depending on which module has been loaded.

working_directory represents the working directory, in which PROCESS will be executed, in case this is supposed to be different to the current directory which can be helpful when executing several runs with slightly different setups.

pseudorandom_seed is the value of the seed for the random number generator. It can be any integer value. If it is not specified, its default value is taken from the system clock.

Other parameters that can be specified in the config section are Niter and factor. Both do not typically need to be changed by the user. Niter is the maximum number of retries that the tool will attempt, if PROCESS fails to find a feasible solution. This means that the tool varies the start values of the iteration variables within a factor given by factor of the original values as this does not change the physical meaning of the input file, but can help the solver to find a better starting point for its iteration. Their default values are Niter=10 and factor=1.5.

Any uncertain parameters should be specified in the “uncertainties” section. Each parameter is specified in its own sub dictionary as shown in the example. For each, the varname and errortype need to be specified as well as the mean and standard deviation std for Gaussian type errors as well as lowerbound and upperbound for Uniform or mean and a percentage for Relative errors. Apart from
a standard Gaussian distribution, also a lower and an upper half Gaussian distribution are available that have a sharp cut off at the mean. Please note, that all distributions are being cut off at the boundaries for the input values for PROCESS! At least one uncertain parameter has to be specified for the program to run and technically there is no upper limit as to how many uncertain parameters can be used. However, for large numbers of uncertain parameters it is recommended to increase the number of sampling points.

There are a number of other parameters in the configuration file that can be specified:

- **output_vars** is a list of strings of output variable names in the MFILE.DAT. These are the variables saved. This list is empty by default and it is therefore crucial for the user to specify the variables of interest because otherwise the tool will not run.

- **no_samples** sets the number of sample points in the Monte Carlo method. It is by default set to its recommended minimum value of 1000, but the user should contemplate higher values especially if a large number of uncertain parameters is involved.

Two parameters that can be further set in the config file (but are not recommended to be changed) are the number of scan points **no_scans** which is by default 5 and the number of allowed unfeasible points in a scan **no_allowed_unfeasible** which is 2 as recommended in [83].

As the distributions of the uncertainties do not have to be represented by a simple Gaussian, reducing the output to two simple numbers like a mean and a standard deviation is not typically possible. Therefore, we have decided to keep all sampled points in the output files. However, to reduce the amount of data we have decided to only store the user specified parameters in the form of a NetCDF binary file. Given that a scan is performed at each sample point, only the last of these scan points is ever kept for evaluation. (There is an option for developers to keep all the data for debugging purposes. However, this should typically not be used in production runs.) These files can be visualised using the **display_uncertainties.py** tool.

### 6.1.14.2 display_uncertainties.py

This is a utility to display the output file uncertainties.nc created by the evaluate_uncertainties.py tool described above.

By default, if run in the working directory of an uncertainty evaluation, it creates a scatter plot of each user defined output parameter against the next parameter in the list. It also shows the 1D histograms of each parameter distribution. If two specific variables are given as arguments, the tool plots only these two against each other.

**Input:** uncertainties.nc

**Output:** Uncertainties_varname1_varname2.pdf

**Usage:**

display_uncertainties.py [-h] [-f FILENAME] [-e END] [v [v ...]]

Program to display uncertainties of a given PROCESS design point.

**positional arguments:**

- **v**
  list of variables to be plotted; default = all
optional arguments:
-h, --help show this help message and exit
-f FILENAME, --filename FILENAME
        uncertainties data file, default = uncertainties.nc
-e END, --end END file format default = .pdf

6.1.15 create_dicts.py

This automatically generates the process_dicts.py file used by PROCESS utility programs. It does this by scanning the Fortran source code. The standard output should be redirected, using

create_dicts.py > process_dicts.py

6.1.16 Batch Jobs

As PROCESS typically runs very fast and does not produce much data output, it is not typically necessary to submit PROCESS runs or any of the python executables as a batch job to the fusion linux machines. However, the evaluate_uncertainties.py tool is one example of a python tool for PROCESS that does run for a long time and does create a lots of output. As the CPU time limit for any non-batch jobs on the fuslw machines is 30 Minutes, any decently sampled evaluate_uncertainties.py run will need to be submitted as a batch job. (Please note, that if you have forgotten to submit your job as a batch job, your job will be terminated after 30 CPU minutes, but as the output is written to file continuously, you should not lose any of the output that has been produced until then.)

To submit a batch job, first create a file called e.g. myjob.cmd. It should contain the following content

```
# @ executable = evaluate_uncertainties.py
# @ arguments
# @ input = /dev/null
# @ output = /ABSOLUTE_PATH_TO_WORK_DIR/ll.out
# @ error = /ABSOLUTE_PATH_TO_WORK_DIR/ll.err
# @ initialdir = /ABSOLUTE_PATH_TO_WORK_DIR/
# @ notify_user = USERNAME
# @ notification = complete
# @ queue
module use /home/PROCESS/modules
module swap python
module load process/master
evaluate_uncertainties.py
```

Once you have created the file you can submit a batch job by typing

llsubmit myjob.cmd

While your job is running you can keep track of its progress by typing llq or xloadl. More information and help with troubleshooting can be found under http://fusweb1.fusion.ccfe.ac.uk/computing/funfaq/ll/
As the uncertainties.nc output file can get quite large, it might be indicated to write the output to the /scratch or /tmp directories as they have faster I/O. Please remember to copy your results into your home directory afterwards, as these directories are not backed up and will be frequently cleaned.

### 6.1.17 mfile_comparison.py

Tool for comparing two MFILEs and outputting significant differences in numerical values.

**Arguments**

- `-f` Files to compare  
- `-s` Save output to file called comp.txt  
- `--acc` Percentage difference threshold for reporting  
- `--verbose` Additional output

### 6.1.18 test_suite.py

The PROCESS test suite allows PROCESS developers to quickly test new modifications to PROCESS using a defined library of reference cases. The test suite is bundled with PROCESS and is a single Python script (*test_suite.py*) with a number of options.

The test suite will provide the following output:

- Summary to terminal and file (*summary.log*),
- PROCESS terminal output log to file for each test case (*run.log*),
- PROCESS error log to terminal and to file if errors for each test case (*error.log*),
- New MFILE.DATs and OUT.DATs for each test case (*new.MFILE.DAT* and *new.OUT.DAT*).

#### 6.1.18.1 Folder Structure

The PROCESS folder structure is shown below. The test suite folder (*test_suite*) is in the main directory with the FORTRAN files. Inside the test suite folder there is the main code (*test_suite.py*) as well as the folders containing the reference cases (*test_files*). The figure below shows the folder structure for the test_suite.

#### 6.1.18.2 Adding a reference case

To add a reference case if you have a local PROCESS repository:

1. Have an IN.DAT that works with the current version of PROCESS
2. Run PROCESS to create an OUT.DAT and MFILE.DAT
3. Create a folder in the location `/my_develop/test_suite/test_files/[test_name]` with a suitable name for the test case (if your local PROCESS folder is `my_develop`).
4. Copy your IN.DAT, OUT.DAT and MFILE.DAT into this folder
5. Copy IN.DAT as \texttt{ref.IN.DAT}

6. Rename OUT.DAT as \texttt{ref.OUT.DAT}

7. Rename MFILE.DAT as \texttt{ref.MFILE.DAT}

8. \texttt{cd ../../} to return to test\_suite folder

9. \texttt{test\_suite.py} to run the test suite to check it works

10. Commit to Git locally and push to the central repository (see instructions in PROCESS manual)

Figure 6.1: PROCESS Test suite folder structure after a test run. The test\_area folder is where the output of the test run is stored. Each test case has a README file which outlines the reasoning for the test.
6.1.18.3 Running the Test Suite

To run the test suite the user should go to the folder `/process/test_suite/` and then run the Python file `test_suite.py`. The test suite Python script has the following optional arguments.

```
python test_suite.py [options]
-h -help  show usage
-s --save save test output to folder with PROCESS revision number in it(e.g. test_r383)
-d --diff [val] set allowed difference between reference case and new output in percentage terms.
-r --ref choose reference folder to use for the test cases default is "test_files"
--debug run reference cases prefixed with "error_" for testing of the error handling in the test suite
```

Examples

```
python test_suite.py -d 10 --save
python test_suite.py -d 1
```

6.1.18.4 Test Suite Output Formatting

`summary.log`

The summary.log file mirrors what was displayed to the terminal during a test suite run. It will look something like the following.

```
PROCESS Test Suite
Date: 2015-12-03
Diff set to: 5.0%
PROCESS Test Cases
Test ==> DEMO1_a31_06_2015 DIFF(6)
Test ==> costs_paper OK
Test ==> test 3 ERROR
Test ==> test 4 OK
PROCESS version r[version] test run complete
```
diff.log

This file will contain the differences between the reference case and the new run for a given test. The file will look like the example below.

PROCESS Test Suite

Test Case: [test_name]
Difference value: [diff]

Differences above allowed value:

<table>
<thead>
<tr>
<th>var</th>
<th>ref</th>
<th>new</th>
<th>%</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>10</td>
<td>15</td>
<td>50</td>
</tr>
<tr>
<td>b</td>
<td>10</td>
<td>9</td>
<td>10</td>
</tr>
</tbody>
</table>

... ... ... ... ...

Variables in ref NOT IN new:

| var 1 | var 2 | ...

Variables in new NOT IN ref:

| var 1 | var 2 | ...

run.log

Run.log will take the output to the terminal for each PROCESS test case and dump it to a file in the test folder in /process/test_suite/test_area/[test_name]/. If there is an error while running PROCESS this will be highlighted in summary.log and then the user can inspect the output in run.log

new.MFILE.DAT

This is the MFILE.DAT the test suite created when running a given test case. After running the test suite will move this file to the folder /process/test_suite/test_area/[test_name]/.
new.OUT.DAT

This is the OUT.DAT the test suite created when running a given test case. After running the test suite will move this file to the folder /process/test_suite/test_area/[test_name]/.

6.2 Python Libraries

All library modules and functions are documented using docstrings. These can be accessed by reading the code directly or via the help() function in Python.

6.2.1 in_dat.py

A set of Python classes to read, modify and write an IN.DAT file.

INVariable(name, value, comment="") Initialises an IN.DAT variable class which requires a name and a value.

INModule(name) Initialises an IN.DAT module class which requires a name. This class stores information for an IN.DAT file which has modules separated with lines with $MODULE_NAME and $END.

INModule.add_variable(var) Adds an INVariable object to the INModule class.

INModule.remove_variable(variable_name) Removes an INVariable object from the INModule class.

INModule.add_line(line) Adds a line from the IN.DAT that isn’t a variable line (e.g. a comment) to the INModule class.

INModule.get_var(var_name) Returns an INVariable object from the INModule class.

INModule.add_constraint_eqn(eqn_number) Adds constraint equation eqn_number to the list of constraint equations InVariable class if the equation is not already in the list.

INModule.remove_constraint_eqn(eqn_number) Removes constraint equation eqn_number from the list of constraint equations InVariable class if the equation is already in the list.

INModule.add_iteration_variable(var_number) Adds iteration variable var_number to the list of iteration variables InVariable class if the variable is not already in the list.

INModule.remove_iteration_variable(var_number) Removes iteration variable var_number to the list of iteration variables InVariable class if the variable is already in the list.

INDATClassic(filename="IN.DAT") Initialises an IN.DAT class which can be given a different filename. This class is used for an IN.DAT with a module structure.
Chapter 6  Utility Programs

INDATClassic.read_in_dat()  Reads an IN.DAT file with modular structure.

INDATClassic.write_in_dat(filename="new_IN.DAT")  Writes a new IN.DAT with modular structure.

INDATNew.read_in_dat()  Reads an IN.DAT file without modular structure.

INDATNew.write_in_dat(filename="new_IN.DAT")  Writes a new IN.DAT without modular structure.

INDATNew.add_variable(var)  Adds an INVariable object to the INDATNew class.

INDATNew.remove_variable(var_name)  Removes an INVariable object from the INDATNew class.

INDATNew.add_constraint_eqn(eqn_number)  Adds constraint equation eqn_number to the list of constraint equations InVariable class if the equation is not already in the list.

INDATNew.remove_constraint_eqn(eqn_number)  Removes constraint equation eqn_number from the list of constraint equations InVariable class if the equation is already in the list.

INDATNew.add_iteration_variable(var_number)  Adds iteration variable var_number to the list of iteration variables InVariable class if the variable is not already in the list.

INDATNew.remove_iteration_variable(var_number)  Removes iteration variable var_number to the list of iteration variables InVariable class if the variable is already in the list.

clear_lines(lines)  Removes comment only lines and replaces multiple empty lines with a single empty line.

variable_type(var_name, var_value)  Checks the type of an IN.DAT variable using DICT_VAR_TYPE

fortran_python_scientific(var_value)  Changes from FORTRAN double precision notation D to Python's e notation.

6.2.2  mfile.py

A set of Python classes to read and extract data from MFILE.DAT.

MFileVariable(var_name, var_description)  Object to contain information for a single MFILE.DAT variable.

MFileVariable.set_scan(scan_number, scan_value)  Sets the variable value for scan number scan_number

MFileVariable.get_scan(scan_number)  Returns the variable value for scan number scan_number

MFileVariable.get_scans()  Returns the variable value for all scans.
Chapter 6  Utility Programs

MFile(filename="MFILE.DAT") Object to contain information for all variables in an MFILE.DAT for all scans.

MFile.search_keys(variable) Search for an MFILE variable in the data dictionary.

MFile.search_des(description) Search for an MFILE description in the data dictionary.

MFile.make_plot_dat(custom_keys, filename="make_plot_dat.out", file_format="row")
Create a PLOT.DAT equivalent for the variables in custom_keys in either row or column format.

MFile.get_num_scans() Returns the number of scans in the MFILE.DAT

read_mplot_conf(filename="make_plot_dat.conf") Reads the config file make_plot_dat.conf and fills the list custom_keys with these variables.

write_mplot_conf(filename="make_plot_dat.conf") Writes a new make_plot_dat.conf adding any additional variables that the user gave make_plot_dat.py at runtime.

6.2.3  process_funcs.py

This library contains a collection of functions used by various PROCESS utilities.

get_neqns_itervars(wdir=’.’) Returns the number of equations and a list of variable names of all iteration variables.

update_ixc_bounds(wdir='.') Updates the lower and upper bounds in DICT_IXC_BOUNDS from IN.DAT.

get_variable_range(itervars, factor, wdir='.') Returns the lower and upper bounds of the variable range for each iteration variable.

itervars: string list of all iteration variable names
factor: defines the variation range for non-f-values by setting them to value * factor and value / factor respectively while taking their PROCESS bounds into account.

For f-values the allowed range is equal to their PROCESS bounds.

cHECK_logfile(logfile='process.log') Checks the log file of the PROCESS output. Stops if an error occurred that needs to be fixed before rerunning.

process_stopped(logfile='process.log') Checks the PROCESS logfile whether it has prematurely stopped.

process_warnings(logfile='process.log') Checks the PROCESS logfile whether any warnings have occurred.

mfile_exists() Checks whether MFILE.DAT exists.

no_unfeasible_mfile(wdir='.') Returns the number of unfeasible points in a scan in MFILE.DAT.

no_unfeasible_outdat(wdir='.') Returns the number of unfeasible points in a scan in OUT.DAT.
vary_iteration_variables(itervars, lbs, ubs) Changes the iteration variables in IN.DAT within given bounds.

- **itervars**: string list of all iteration variable names
- **lbs**: float list of lower bounds for variables
- **ubs**: float list of upper bounds for variables

get_solution_from_mfile(neqns, nvars, wdir='.') Returns:

- **ifail**: error value of PROCESS
- the objective functions
- the square root of the sum of the squares of the constraints
- a list of the final iteration variable values
- a list of the final constraint residue values
- If the run was a scan, the values of the last scan point will be returned.

get_solution_from_outdat(neqns, nvars) Returns:

- **ifail**: error value of PROCESS
- the objective functions
- the square root of the sum of the squares of the constraints
- a list of the final iteration variable values
- a list of the final constraint residue values
- If the run was a scan, the values of the last scan point will be returned.

### 6.2.4 process_config.py

A collection of Python classes for configuration files used by various PROCESS utilities, e.g. run_process.py or test_process.py. It contains a base class `ProcessConfig` and two derived classes `RunProcessConfig` and `TestProcessConfig`.

`ProcessConfig()` Object that contains the configuration parameters for PROCESS runs.

- **filename**: Configuration file name
- **wdir**: Working directory
- **or_in_dat**: Original IN.DAT file
- **process**: PROCESS binary
- **niter**: (Maximum) number of iterations
- **u_seed**: User specified seed value for the random number generator
- **factor**: Multiplication factor adjusting the range in which the original iteration variables should be varied
- **comment**: Additional comment to be written into README.txt

`ProcessConfig.echo_base()` Echos the attributes of the base class to standard output.

`ProcessConfig.echo()` Echos the values of the current class to standard output.
Chapter 6 Utility Programs

ProcessConfig.prepare_wdir() Prepares the work directory for the run.

ProcessConfig.create_readme(directory='.') Creates a file called README.txt containing ProcessConfig.comment.

ProcessConfig.modify_in_dat() Modifies the original IN.DAT file.

ProcessConfig.setup() Sets up the program for running.

ProcessConfig.run_process() Runs PROCESS binary.

ProcessConfig.get_comment() Gets the comment line from the configuration file.

ProcessConfig.get_attribute(attribute) Gets a class attribute from the configuration file.

ProcessConfig.set_base_attributes() Sets the attributes of the base class.

TestProcessConfig(filename='test_process.conf') Object that contains the configuration parameter of the test_process.py program.

ioptimz : sets ioptimz (optimisation solver) in IN.DAT.

epsvmcm : sets epsvmc (VMCON error tolerance) in IN.DAT.

epsfcm : sets epsfmc (finite diff. steplength) in IN.DAT.

minmax : sets minmax (figure of merit switch) in IN.DAT.

TestProcessConfig.echo() Echoes the values of the current class to std out.

TestProcessConfig.modify_in_dat() Modifies IN.DAT using the configuration parameters.

RunProcessConfig(filename='run_process.conf') Configuration parameters of the run_process.py program.

no_allowed_unfeasible : The number of allowed unfeasible points in a sweep

create_itervar_diff : Boolean to indicate the creation of a summary file of the iteration variable values at each stage

add_ixc : List of iteration variables to be added to IN.DAT.

del_ixc : List of iteration variables to be deleted from IN.DAT.

add_icc : List of constrained equations to be added to IN.DAT.

del_icc : List of constrained equations to be deleted from IN.DAT.

dictvar : Dictionary mapping variable name to new value (replaces old or gets appended)

del_var : List of variables to be deleted from IN.DAT.

RunProcessConfig.get_attribute_csv_list(attribute) Get a class attribute list from the configuration file; expects comma separated values.

RunProcessConfig.set_del_var() Sets the RunProcessConfig.del_var attribute from the config file.
RunProcessConfig.set_dictvar() Sets the RunProcessConfig.dictvar attribute from config file.

RunProcessConfig.echo() Echoes the values of the current class.

RunProcessConfig.modify_in_dat() Modifies IN.DAT using the configuration parameters.

RunProcessConfig.modify_vars() Modifies IN.DAT by adding, deleting and modifying variables.

RunProcessConfig.modify_ixc() Modifies the array of iteration variables in IN.DAT.

RunProcessConfig.modify_icc() Modifies the array of constraint equations in IN.DAT.

6.2.5 process_dicts.py

A collection of dictionaries and lists used by various PROCESS utilities.

IFAIL_SUCCESS This is the PROCESS error code of a successful run.

PARAMETER_DEFAULTS Default values for making a PLOT.DAT file from MFILE.DAT.

DICT_VAR_TYPE Maps the PROCESS variable name to its value type. The value type can be one of int_array, int_variable, real_array or real_variable.

DICT_IXC_SIMPLE Maps the string number of the iteration variable to its variable name.

DICT_IXC_FULL Maps the string number of the iteration variable to a dictionary that contains the variable name under 'name', the default lower variable bound (float) under 'lb' and the default upper variable bound (float) under 'ub'.

DICT_IXC_BOUNDS Maps each iteration variable name to a dictionary that contains the default lower variable bound (float) under 'lb' and the default upper variable bound (float) under 'ub'.

NON_F_VALUES List of iteration variable names that start with an f, but are not f-values.

DICT_NSWEEPS2IXC Maps the sweep variable number nsweep to the respective iteration variable number, if applicable.

DICT_IX2NSWEEPC Maps the iteration variable number to the respective sweep variable number nsweep, if applicable.

DICT_TF_TYPE Maps the PROCESS TF coil type number to its type.

DICT_OPTIMISATION_VARS Maps each figure of merit number to its description.

DICT_IXC_DEFAULT Maps each iteration variable name to its default value.

Add new stuff used by GUI etc.

6.2.6 a_to_b_config.py

To do...
6.2.7 proc_plot_func.py

A collection of functions and lists used by plot.proc.py.

RADIAL_BUILD A list of radial build variables.
VERTICAL_BUILD A list of vertical build variables.
FILL_COLS A list of plotting colours

For all of the functions below the arguments are:
  axis : Matplotlib axis object to plot to
  mfile_data : MFILE data object MFile
  scan : Scan number to plot

plot_plasma(axis, mfile_data, scan) Function to plot plasma
poloidal_cross_section(axis, mfile_data, scan) Function to plot machine build
plot_tf_coils(axis, mfile_data, scan) Function to plot the TF coils
plot_pf_coils(axis, mfile_data, scan) Function to plot the PF coils
plot_geometry_info(axis, mfile_data, scan) Function to plot the geometry info block
plot_physics_info(axis, mfile_data, scan) Function to plot the physics info block
plot_magnetics_info(axis, mfile_data, scan) Function to plot the magnetics info block
plot_power_info(axis, mfile_data, scan) Function to plot the power info block
plot_current_drive_info(axis, mfile_data, scan) Function to plot the current drive info block
plot_geometry_info(axis, mfile_data, scan) Function to plot the geometry info block

6.2.8 diagnose_funcs.py

A collection of functions used by diagnose_process.py.

plot_normalised_ixc(mfilename='MFILE.DAT') Plots the normalised values of the iteration variables of a given MFILE.DAT.
plot_normalised_icc_res(mfilename='MFILE.DAT') Plots the normalised values of the constraint residuals of a given MFILE.DAT.

6.3 User Interface

The user interface is still being developed, but currently allows for viewing and editing of IN.DAT files in a web browser. It can be found in the utilities/gui directory. See Chapter 4.2 for details.
Chapter 7

Code Management Tools

This chapter will be of interest to people involved in the continuing maintenance of the PROCESS source code. As stated elsewhere, the source code is maintained by CCFE, and resides in a Git repository on the CCFE servers.

7.1 The Makefile

In addition to its normal role for compilation, the makefile (named Makefile) includes a number of utility functions that perform tasks such as automatic generation of the code documentation, and the creation of a tar file containing the entire source code, its documentation files, and the input and output files. This has proved to be of great benefit in keeping all of the data from a given run together for archival purposes.

7.1.1 Compilation

Compilation is trivially performed using the makefile. Currently, the available options are (type the following on the command line):

```
make ARCH=FUN          CCFE Fusion Unix Network (Intel Fortran ifort compiler)
make ARCH=GFORT        GNU Fortran compiler (gfortran; N.B. only versions 4.6.3 or above)
make ARCH=JAC          JET Analysis Cluster (pgf95 compiler)
```

The Intel Fortran compiler is the default option, so typing simply `make` will have the same effect as typing `make ARCH=FUN`.

The code is almost trivial to port to new architectures, by adding extra stanzas to the makefile as required.

To force a full recompilation, type `make clean`. N.B. This will remove (without prompting) all “backup” files (*.~) as well as certain other files from the working directory.

Extra run-time diagnostics (array-bound checking, etc.) is turned on by default at present; the code is sufficiently quick to run that the performance penalty is barely noticeable. However, this can be turned off (beware...results can change!), by adding `DEBUG=NO` to the relevant command above, i.e. type `make ARCH=... DEBUG=NO`. N.B. there must be no spaces either side of the = characters in any of the above commands.
Chapter 7  Code Management Tools

7.1.2 Archiving utilities

The makefile can be used in a number of ways to pack the various file sets together into a single compressed tar file, for ease of portability and archiving.

- **make tar**: Typing this command produces a file `process.tar.gz` that contains all of the source files, utility programs and documentation files necessary to run the program from scratch on a new machine or in a new directory. (Note that the input files `IN.DAT` and `device.dat` are not included.) This is useful for transferring new copies of source files, etc. into an existing directory already containing a previous `PROCESS` run, as the pre-existing customised input files will not be overwritten. To extract the individual files again, copy the file to the destination working directory and type `tar zxvf process.tar.gz`

- **make archive**: Typing this command produces a file `process_run.tar.gz` containing the working directory’s input and output files (`IN.DAT`, `OUT.DAT`, `PLOT.DAT`, `MFILE.DAT` and `device.dat`) (which must exist to avoid an error message; see Section 3.0.1), together with all of the source files, utility programs and documentation files. These files together comprise the full set that define a given run, and so the file produced by this command is suitable for long-term storage for archival purposes. To extract the contents, type `tar zxvf process_run.tar.gz`

Note that each of the above commands will overwrite the given named tar file if one already exists in the working directory.

7.1.3 Code documentation

The makefile can also be used to produce source code documentation, as described in the next section.

7.2 Automatic Documentation

The `PROCESS` source code is self-documenting to a degree, using an included parser program (`autodoc`) to generate html files for each subprogram from specially-formatted comment lines within the code. It is the responsibility of the programmer to keep the `autodoc` comments within the source code relevant, comprehensive and up-to-date! Use the examples in the code as a starting basis for new routines; the output section corresponding to the various `autodoc` tags should be self-explanatory. See also Section 5.1.

The following files are used:

```
autodoc.f90
adheader.src
adfooter.src
```

To create the (~ 430) html files from the source code, type `make html`. Then, point your favourite web browser to the file `process.html` or `calling_tree.html` in the working directory. The variable descriptor file `vardes.html` (see Section 2.2) is also produced at the same time (though it may be helpful to modify the file manually afterwards to remove empty sections from it).

In addition, a full `LaTeX` User Guide (this document!), contained within `process.tex` and the other `.tex` files called from it, is rigorously maintained to ensure its continuing strict agreement with
the evolving source code. To create an Adobe PDF file `process.pdf` from `process.tex` (and the associated PostScript figures), type `make guide`.

Finally, to produce both the html files and the User Guide in one go, simply type `make doc`.

### 7.3 Code Updates and Release Procedure

This section describes the procedures that should be followed whenever new commits to the `develop` or `master` branches of the `PROCESS` Git repository are to be made, and how new code releases are performed. It is assumed that readers have a working knowledge of Git commands.

#### 7.3.1 Initial access to the source code

To gain access to the `PROCESS` source code Git repository, you need to be given permission to do so via the CCFE GitLab server.

1. Use a web browser to go to `http://git.ccfe.ac.uk`
2. Login using your normal CCFE computer login details.
3. Assuming this is successful, contact the GitLab `PROCESS` “Owner” (currently James Morris [james.morris2@ccfe.ac.uk](mailto:james.morris2@ccfe.ac.uk)), who will add you to GitLab as a `PROCESS` “Developer”.
4. (If this is your first access to GitLab, you may have to set up SSH keys and a few other things; follow the instructions on screen.)
5. Login to a Fusion Unix Network machine.
6. `cd` `SomewhereAppropriate` (you choose!)
7. `git clone git@git.ccfe.ac.uk:process/process.git -b develop my_develop` This copies the `develop` branch of the repository into a local folder `my_develop`, which will be created if it does not already exist.
8. `cd my_develop`
9. `git checkout develop`
   The sequence of commands above will provide you with a full copy of the `develop` branch of the `PROCESS` source code, Python utilities and all the documentation files.
10. `make all`
   This performs a full compilation and builds the executable, creates the User Guide, creates the python dictionaries and produces all the web-based documentation.

#### 7.3.2 Git workflow

The code management methodology is based on the so-called “gitflow” workflow. There are two main branches:

- `develop`, which is the basis for all code development work, and changes are committed to it frequently.
• master, which contains official “release” versions of PROCESS, and is updated on a roughly three-monthly timescale.

Development work on new models should use separate branches, named e.g. dev_mymodel, split from the develop branch. It is a very good idea to merge the develop branch into dev_mymodel frequently during the course of the work, so that changes to the main branch are transferred to the new model’s files with minimal effort.

Once the new model has been finished and tested successfully (complete with full documentation — see Section 5.1), the branch should be merged back into the develop branch.

7.3.2.1 Branching from develop

The following commands create a new branch in which to work on a new model. We assume that you are already in the process directory created above (Section 7.3.1).

1. git checkout develop (just to ensure that this is the branch from which you will be branching)
2. git branch dev_mymodel
3. git checkout dev_mymodel
4. git push origin HEAD (updates the central GitLab repository)

7.3.2.2 Working on a new model

Edit your local copies of the files as necessary. Whenever you want to save the changes back to the repository (it is prudent to do this at the end of each working day, as well as when the changes are complete, the code compiles and all test runs are successful!), use the following commands:

1. git status (this will show a list of modified files)
2. git add changedfile1 changedfile2 ...
   (This ‘stages’ the modified files marking them as ready for committing).
   git commit
   Alternatively, just use
   git commit -a
   This commits all modified tracked files.
3. An editor window will open; add a line summarising the changes you have made, save and close the window. This will initiate the commit to your local copy of the repository.
4. git push
   Copies the changes you have made locally to the version in the central GitLab repository. This uses a merging process, but if no-one else has changed your branch then the central version will simply become a copy of your local version.

7.3.2.3 Merging develop into working branch

It is a good idea to periodically merge the develop branch into the branch in which you are working, to ensure that any changes made in develop are included in your working branch.
1. Firstly, make sure you have committed your latest changes into the central GitLab repository as described in the previous section. Set the directory containing the working branch as your current directory.

2. `git pull`.
   Merges the version in the central repository into the local branch by copying over any changes that have been made in the version stored centrally.

3. `git checkout develop`
   This switches the "current branch" to `develop`.

4. `git pull`
   Updates the current branch of the local repository to the same branch on the central repository.

5. `git checkout dev_mynewmodel`
   This switches the "current branch" to `dev_mynewmodel`.

6. `git merge develop`
   Merges `develop` into the `dev_mynewmodel` branch.

7. Look for messages on the screen containing the word "conflict" indicating that some files cannot be merged directly. This typically happens if the same (or very closely-spaced) lines have been edited in both the `develop` and `dev_mynewmodel` branches.
   If any files are affected, they will be listed. Edit them and look for any lines containing ======.
   Such lines separate the changes made in the two branches, as in:
   
   `<<<<<< HEAD
   This line was edited in dev_mynewmodel branch
   ======
   This line was edited in develop branch
   >>>>>> develop`

   Resolve the conflict(s) as necessary. Then type `git add file1 file2 ...`, where `file1` etc. are the names of the files you removed conflicts from. Finally type `git commit` and edit the change log file.

8. `git push`
   Updates the central repository.

### 7.3.2.4 Merging working branch back into `develop`

Once the work on a model has been completed, fully tested and documented, the working branch should be merged back into the `develop` branch. This is done almost exactly as in the previous section, but there are a few crucial differences:

1. Firstly, make sure you have committed your latest changes to the working branch into the central GitLab repository.

2. `git pull`
   Merges the version in the central repository into the local branch by copying over any changes that have been made in the version stored centrally.

3. `git checkout dev_mynewmodel`
4. git checkout develop
5. git merge dev_mynewmodel
6. Look for any conflicts and resolve them as described in the previous section.
7. git push

### 7.3.2.5 Committing changes to develop

Whenever a commit to the `develop` branch is to be made, the following procedure should be followed. Ensure all documentation is up to date (see Section 5.1).

1. In routine `inform` of file `process.f90`, change the definition of `progver` by incrementing the revision number by one to `XYZ` (for instance) and the Release Date. It is important to keep exactly the same format.
2. Add a brief comment to the bottom of source file `process.f90` describing the changes made since the last commit in the same branch. Start the line with `! GIT XYZ:`, following the existing examples.
3. If any of the User Guide `.tex` files have been modified, edit the definition of `\version` in `process.tex` by changing the Revision (to `XYZ`) and the date.
4. If you have changed any "use" statements in the code, or any compilation dependencies in the `Makefile`, run `make clean` then `make all` This ensures that all the code and documentation compiles successfully.
5. `make all`
   This ensures that all the code and documentation compiles successfully.
6. Run the input file(s) in the `tests` folder to ensure PROCESS runs correctly.
7. Close all open editor windows. The commit will not work otherwise.
8. `git commit -a`
   This commits all modified tracked files.
   Alternatively, you can this in two steps:
   
   git add process.f90 process.tex changedfile1 changedfile2 ...
   (This ‘stages’ the modified files marking them as ready for committing).
   git commit
9. An editor window will open; add a line summarising the changes you have made. Use a format like this:

   `rXYZ A summary of the changes made`

   Further details. Changes due to Git issues can be described like this:
   #270 Description
   #273 Description

   Save and close the window. This will initiate the commit to your `local` copy of the repository.
10. `git tag -a rXYZ -m ‘Revision XYZ’`
11. git push
12. git push origin rXYZ

The instructions given in Section [7.3.3] should now be followed to make the new develop release available to all users.

### 7.3.2.6  Merging develop into master

(to follow)

### 7.3.3  Full code rebuild

The standard PROCESS executables and the corresponding documentation available to all users are stored in the functional account called PROCESS on the CCFE Fusion Unix Network. Whenever the master or develop branches are updated a full rebuild of the standard executables and documentation should be performed. This is done as follows:

1. Ensure that all key users have been informed (using the commit message described above or directly).
2. alter PROCESS (this changes your current login to that of the PROCESS user; only registered individuals are able to do this)
3. cd develop (or cd master, as appropriate to the branch to be rebuilt)
4. git pull
   Updates the version stored in this folder to the version stored centrally.
5. make all This is essential because the Git repository does not include any of the files generated in the compilation process.
6. exit (to return to your own username again)

The make all step performs the rebuild of the process.exe executable file, updates the User Guide and all the html files, and recreates the Python dictionaries as required by the Python utilities.
Chapter 8

Acknowledgements & Bibliography

The authors would like to thank the following people for many useful and revealing discussions during their work on PROCESS:

— John D. Galambos, Paul C. Shipe and Y-K. Martin Peng of Oak Ridge National Laboratory, USA;
— Ian Cook, Robin Forrest, Winston Han, Roger Hancox and Panos Karditsas, all formerly of Fusion Physics Department, UKAEA Fusion;
— John Hicks, formerly of Engineering Department, UKAEA Fusion;
— Chris Gardner, formerly of Microwave and Interpretation Department, UKAEA Fusion;
— Tim Hender of CCFE, and all the co-authors of reference [17];

This work was funded by the RCUK Energy Programme under grant EP/I501045 and the European Communities under the contract of Association between EURATOM and CCFE. The views and opinions expressed herein do not necessarily reflect those of the European Commission. Part of this work was carried out within the framework of the European Fusion Development Agreement.
Bibliography


[27] H. R. Wilson, Nuclear Fusion 32 (1992) 257


[38] “Pulsed Fusion Reactor Study”, AEA Fusion Report AEA FUS 205 (1992)


[40] F. Warmer, “Stellarator Plasma Geometry model for the systems code PROCESS”, IPP Greifswald, Germany, internal note, 19/06/2013

[41] F. Warmer, “Stellarator Divertor model for the systems code PROCESS”, IPP Greifswald, Germany, internal note, 21/06/2013


[51] F. Warmer, “Stellarator Modular Coil model for the systems code PROCESS”, IPP Greifswald, Germany, internal note, 31/07/2013


[63] P. J. Knight, “PROCESS IFE Build Details”, F/MI/PJK/LOGBOOK12, pp. 52, 53, 56, 57


https://idm.euro-fusion.org/?uid=2M94N2\&version=v1.0\&action=get_document
Appendix A

The Optimisation Solver Explained

To give the user a better understanding of the optimisation solver implemented in PROCESS and the interpretation of its results, we give a short introduction into the mathematical background for solving these type of problems as well as the specific algorithm used.

In section A.1 the general nonlinear programming problem is defined, which is the mathematical description of our problem. This problem is typically formulated using Lagrange multipliers (c.f. A.2) and is solved numerically most efficiently using sequential quadratic programming (c.f. A.3). The Fortran subroutine that is used to implement such an optimisation solver in PROCESS is called VMCON (c.f. A.4), which iterates between solving a local quadratic subproblem (c.f. A.5) and a one dimensional line search (c.f. A.6). As the method uses a quasi-Newtonian approach the Hessian matrix is approximated by a variant of the Broyden-Fletcher-Goldfarb-Shanno update (A.7). Section A.8 summarises the symbol convention used in this chapter.

A.1 The General Nonlinear Programming Problem

Mathematically the general nonlinear programming problem or nonlinear constrained optimisation problem is defined as

\[
\begin{align*}
\text{minimise} & \quad f(x), \\
\text{subject to} & \quad c_i(x) = 0, \quad i = 1, \ldots, k, \\
& \quad c_i(x) \geq 0, \quad i = k + 1, \ldots, m,
\end{align*}
\]

where both the objective function \( f(x) \) and the constraints \( c_i(x) \) are nonlinear functions of the \( n \)-dimensional vector of variables \( x \) with bounds \( x \in \Omega \). In this context, all \( x \in \Omega \) that fulfill the constraints \( c_i(x) \) are called feasible. They describe the allowed space in which we are trying to optimise the objective function \( f(x) \). Note that any maximisation problem can be written as a minimisation by using \( f_{\text{new}}(x) = -f(x) \) and that any equality constraint \( c(x) = a \) can be rewritten as \( c_{\text{new}}(x) = c(x) - a = 0 \). Any inequality constraint can therefore be rearranged analogously to fit the form described in eq. A.1c.
Figure A.1: Illustration of Lagrange Multiplier Method (credit Wikipedia) showing two contour lines of the objective function $f(x, y) = d_i$ (dark blue dashed lines) and the nonlinear constraint $g(x, y) = c$ (red solid line) as well as their gradients (blue and red arrows) at various positions including the constrained optimum (light blue and orange arrows).
A.2 The Lagrange Method

The general nonlinear programming problem can be solved mathematically using Lagrange’s method. It assumes that the constraints cannot be used to explicitly reduce the parameter space of the iteration variables - as it is typically the case for non-linear constraints and objective functions - and is therefore a powerful method applicable to a general class of problems.

If we assume for simplicity that we have a 2D problem with only one equality constraint \( c(x, y) = 0 \), we know that we only need to search for the optimum along that constraint. At the optimum, the value of the objective function will then be stationary, i.e. it does not locally increase or decrease along the constraint. As the gradient of a function is perpendicular to its contour lines of \( f(x, y) = d \), this is equivalent to saying that the gradient of the objective function at that point is parallel to the gradient of the constraints:

\[
\nabla f(x, y) = -\lambda \nabla c(x, y),
\]

(A.2)

where the factor \( \lambda \) is necessary as only the direction, but not the magnitude nor the sign of the gradients need to be equal. This is also illustrated in Figure A.1.

When expanding the method to several equality and inequality constraints we can make use of the Lagrange function. For the nonlinear programming problem described by (A.1) it is given by

\[
L(x, \lambda) = f(x) - \sum_{i=1}^{m} \lambda_i c_i(x),
\]

(A.3)

with the corresponding Lagrangian multipliers \( \lambda_i \). It allows us to formulate the first order necessary conditions for a constrained optimum \( x^* \) with corresponding Lagrange multipliers \( \lambda^* \), the Karush-Kuhn-Tucker (KKT) conditions,

\[
\nabla_x L(x^*, \lambda^*) = \nabla_x f(x^*) - \sum_{i=1}^{m} \lambda_i \nabla_x c_i(x^*) = 0,
\]

(A.4a)

\[
\lambda^*_i c_i(x^*) = 0, \quad i = 1, \ldots, m,
\]

(A.4b)

\[
c_i(x^*) = 0, \quad i = 1, \ldots, k,
\]

(A.4c)

\[
\lambda^*_i \geq 0, \quad i = k + 1, \ldots, m,
\]

(A.4d)

\[
c_i(x^*) \geq 0, \quad i = k + 1, \ldots, m.
\]

(A.4e)

Please note that by construction the Lagrange multipliers \( \lambda^*_i \) fulfilling the KKT conditions are describing the derivative of the objective function with respect to the constraint equation \( \frac{df}{dc_i} \) and are therefore a measure of how much the objective function changes with respect to each constraint.

In the special case of a continuously differentiable convex objective function \( f(x) \) and equality constraints as well as affine inequality constraints, these KKT conditions are also sufficient for a global optimum. The PROCESS optimisation solver has been designed to converge on the KKT conditions, but does not test whether these are sufficient for a global optimum. It is therefore crucial that the user verifies that a global optimum has been found.

Furthermore, these conditions and therefore the solver, assume that both the objective function and constraints are at least first order continuously partially differential, i.e. that their first order partial derivatives are all continuous functions. This might not always be the case in PROCESS and is a potential source of errors.

---

\(^1\) Please note that what is called figure of merit in PROCESS is called objective function in the mathematical optimisation context. Hence, both names are used equivalently in this document.
Appendix A  The Optimisation Solver Explained

A.3 Sequential Quadratic Programming (SQP)

Based on the Lagrange method, sequential (also successive or recursive) quadratic programming is the most efficient method to numerically solve constrained nonlinear optimisation problems \[74\]. It combines a simple solver for a quadratic optimisation problem with linear constraints that determines a search direction \( \delta \) with a line search to find an optimum along that search direction. It is a type of the more general feasible direction methods which is itself a type of primal method that solve nonlinear programming problems in the \( n - m \) dimensional feasible subspace \[75\].

A.4 VMCON

The optimisation solver implemented in PROCESS is the Fortran routine VMCON \[5\]. It is a modified version of the vf02ad routine from the Harwell Software Library\(^2\) and implements a SQP method originally suggested by Powell\(^3\)\[6\] based on work by Han \[77\]. As stated before VMCON is designed to converge on a solution of the necessary KKT conditions \[A.4\] but does not check the sufficient conditions. Its convergence criterion is therefore given by

\[
|\nabla x f(x^{j-1})^T \cdot \delta^j| + \sum_{i=1}^{m} |\lambda_i^j c_i(x^{j-1})| < \text{epsvmc}
\]

where epsvmc is a user specified error tolerance, \( \delta^j \) is the vector in direction of the next line search (c.f. section \[A.6\]) and \( j \) is the counter of the sequential quadratic programming iteration. Hence, the first part estimates the predicted improvement due to the next line search and the second part measures the error in the complimentary condition \[A.4b\] of the KKT conditions.

Figure \[A.2\] describes the flow chart of the VMCON routine, while the various values of the return parameter ifail\(^4\) are described and interpreted in Table \[A.1\]

A.5 The Quadratic Subproblem (QSP)

Within sequential quadratic programming the complex nonlinear problem is broken down into solving a sequence of local quadratic subproblems with linear constraints. This is based on the assumption that locally the problem is well approximated by a second order Taylor expansion of the Lagrange function. Hence, the local quadratic subproblem is described by

\[
\text{minimise} \quad Q(\delta) = f(x^{j-1}) + \delta^T \nabla x f(x^{j-1}) + \frac{1}{2} \delta^T \nabla xx L(x^{j-1}, \lambda^{j-1}) \delta \quad (A.6a)
\]

subject to

\[
\delta^T \nabla x c_i(x^{j-1}) + c_i(x^{j-1}) = 0, \quad i = 1, \ldots, k, \quad (A.6b)
\]

and

\[
\delta^T \nabla x c_i(x^{j-1}) + c_i(x^{j-1}) \geq 0, \quad i = k + 1, \ldots, m, \quad (A.6c)
\]

where \( \delta = x - x^{j-1} \), the index \( j - 1 \) indicates the parameter values of the previous iteration\(^5\) and \( \nabla xx L(x^{j-1}, \lambda^{j-1}) \) is the Hessian of the Lagrange function. The solution of the QSP \( \delta \) describes the change of the current iteration variable vector that minimises the local approximation of the problem. To assure convergence even from bad starting points, it is not directly used to update the iteration

---

\(^2\)www.hsl.rl.ac.uk

\(^3\)This should not be confused with Powell’s algorithm \[76\] which solves a multidimensional unconstrained minimisation problem without derivatives.

\(^4\)Note, within the VMCON routine ifail is called info.

\(^5\)Note j is called nqp in the VMCON routine.
Appendix A  The Optimisation Solver Explained

Setup:
• initialise \( B \)
• evaluate \( f, c, \nabla_x f, \nabla_x c \)
• \( j = 0, \text{nfev}=1 \)

SQP iteration

Solve QSP,
\( j = j + 1 \)

Calculate \( \lambda_j, \nabla_x L, \mu_j \)

Line search

Evaluate \( \Phi(f, c) \)

\( l = 1 \):
• calculate \( \Delta \)
• set \( \alpha = 1 \)

\( l > 1 \):
• test convergence
• calculate \( \alpha \)

update \( x^j \),
evaluate \( f, c, \text{nfev} += 1, l += 1 \)

BFGS update

Figure A.2: This is the flow chart of the VMCON optimisation solver. The criteria for and the interpretation of the successful (\( \text{ifail} = 1 \)) and unsuccessful (\( \text{ifail} \neq 1 \)) return parameters are described in Table [A.1].
<table>
<thead>
<tr>
<th>ifail</th>
<th>Description</th>
<th>Meaning</th>
<th>Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>VMCON: Improper input parameters</td>
<td>The input parameters to the solver are wrong, e.g. negative number of iteration variables.</td>
<td>This needs to be fixed on the developer side and should only occur in the test phase of new modules.</td>
</tr>
<tr>
<td>1</td>
<td>VMCON: Normal return</td>
<td>VMCON has found a solution that fulfills the necessary conditions within the specified error tolerances (c.f. eq. [A.5]).</td>
<td>Test whether the solution is a global solution by using different starting parameters.</td>
</tr>
<tr>
<td>2</td>
<td>Too many function calls</td>
<td>During the line search VMCON has reached the maximum number of total function calls ( \text{maxfev} = 100 ).</td>
<td>VMCON struggles to find a solution. Retry with different start parameters.</td>
</tr>
<tr>
<td>3</td>
<td>Line search required 10 functions calls</td>
<td>The results produced by input objective function/constraints and their gradients are inconsistent. This can be due to numerical noise in the input functions.</td>
<td>The developer needs to check the consistency and numerical robustness of the objective function, constraints and their derivatives. Perhaps the accuracy in numerical integrations/differentiations needs to be higher. As a user, try changing the iteration bounds or adding other iteration variables.</td>
</tr>
<tr>
<td>4</td>
<td>Uphill search direction was calculated</td>
<td>The quadratic subproblem has suggested a search direction in which the objective function only increases.</td>
<td>This happens if an inconsistency between the objective function, the constraints and their respective derivatives occurs (c.f. ifail = 3).</td>
</tr>
<tr>
<td>5</td>
<td>qpsub: No feasible solution or bad approximation of Hessian</td>
<td>Either no optimum lies within the space allowed by the constraints and variable bounds or the identity matrix is not a good first approximation of the Hessian.</td>
<td>As a user, add a new iteration variable, as developer, try using a multiple of the identity matrix as initial Hessian instead.</td>
</tr>
<tr>
<td>6</td>
<td>qpsub: Singular matrix in quadratic subproblem or restriction by artificial bounds</td>
<td>This is fairly self-explanatory.</td>
<td>If this is meaningful, widen the boundaries of the iteration variables.</td>
</tr>
</tbody>
</table>

Table A.1: Summary of the description and meaning of the VMCON return parameters ifail.
variable vector, but describes the direction of the line search in which a 1d function is minimised using a Newtonian method (c.f. section A.6). Being a second order approximation to the original problem, it typically has a faster convergence than sequential linear programming (SLP) methods [78, chap. 10.4].

To allow the applicability of the solver to more general problems, Powell [6] substituted the Hessian \( \nabla_x \nabla \text{L}(x^{j-1}, \lambda^{j-1}) \) with a positive definite approximation \( B \). This means that both the objective function \( f(x) \) and the constraint equations \( c_i(x) \) only have to be continuously differentiable instead of twice continuously differentiable with respect to the iteration variable vector \( x \). This makes the method a Quasi-Newtonian method as opposed to true Newtonian methods. How this approximation is initialised and updated is described in more detail in the section about the Broyden-Fletcher-Goldfarb-Shanno update A.7.

To solve the QSP VMCON uses \texttt{harwp} a modification of the the Harwell library routine \texttt{VE02AD}, which in itself uses the subroutine \texttt{harwp/LA02AD} to find a feasible point within the variable bounds and linearised constraints. Both routines go back to a method by Fletcher [79, 80, 81]. The Lagrange multipliers are also determined from results of the \texttt{harwp} routine.

If the routine cannot find a feasible point it fails with \texttt{ifail} = 5 (c.f. Table A.1). As the routine is only checking the local linear approximation of the constraints rather the full non-linear constraints, there is a chance that a feasible point exists even though the routine fails with \texttt{ifail} = 5. In these cases, it is possible that the first approximation of the Hessian has not been good and the algorithm has, therefore, taken an inappropriately large step. Then using a multiple of the identity matrix will improve convergence of the algorithm.

If a singular matrix is encountered with the QSP solver or the solution is restricted by the artificial bounds, VMCON fails with \texttt{ifail} = 6. In this case it can be helpful to widen the boundaries of the iteration variables.

### A.6 The Line Search

The line search is an essential part of the SQP algorithm. As Powell [6] pointed out, it is necessary to allow convergence from poor starting conditions. It uses the vector \( \delta \) from the QSP to update \( x^j = x^{j-1} + \alpha^j \delta^j \) where the step-length parameter \( \alpha^j > 0 \) is determined by the line search as the parameter that minimises

\[
\Phi(\alpha) = f(x) + \sum_{i=1}^{k} \mu_i |c_i(x)| + \sum_{i=k+1}^{m} \mu_i |\min(0, c_i(x))| \tag{A.7}
\]

where the weights are defined as

\[
\mu_i = \begin{cases} 
|\lambda^1_i| & \text{if } j = 1, \\
\max \left( |\lambda^j_i|, 1/2(\mu_i^{j-1} + |\lambda^j_i|) \right) & \text{if } j > 1
\end{cases} \tag{A.8}
\]

to assure maximally efficient convergence [77]. Note, that in this method locally inactive inequality constraints (c.f. 2.5.2) are not having any effect. It should always be possible, to find a solution that fulfills

\[
\Phi(\alpha = 0) > \Phi(\alpha^j), \tag{A.9}
\]

if

\[
\frac{d\Phi}{d\alpha} \bigg|_{\alpha=0} < 0. \tag{A.10}
\]
In case the derivative is positive \((dflsa \geq 0)\), an uphill search direction has been determined and the code stops with \(ifail = 4\). This typically only happens, if the objective function or constraints are inconsistent with their own derivatives.

As the line search tries to determine the optimum of a one dimensional, but fully non-linear function \(\Phi(\alpha)\), it creates a series of \(\alpha_l\) values. At each iteration \(l\), a quadratic local function \(\Phi_l(\alpha)\) fulfilling the boundary conditions \(\Phi_l(0) = \Phi(0), \Phi_l'(0) = \Delta\) and \(\Phi_l(\alpha_{l-1}) = \Phi(\alpha_{l-1})\) is minimised, where typically \(\Delta = \Phi'(0)\). This leads to

\[
\Phi_l(\alpha) = \Phi(0) + \Delta \alpha + \frac{\Phi(\alpha_{l-1}) - \Phi(0) - \Delta \alpha_{l-1}}{\alpha_{l-1}^2} \alpha^2
\]

and minimising this gives

\[
\alpha_{min} = -\frac{\Delta \alpha_{l-1}^2}{2(\Phi(\alpha_{l-1}) - \Phi(0) - \Delta \alpha_{l-1})}.
\]

Powell then sets \(\alpha_l = \min(\alpha_{min}, 0.1 \alpha_{l-1})\). On each iteration the convergence of the line search is tested. It is reached, if

\[
\Phi(\alpha_l) - \Phi(0) < 0.1 \Delta.
\]

which assures that the change in the function is small in comparison to its derivative and 0.1 is a factor determined by experience. If this criterion is successful, the code exits the line search and updates all function evaluations.

As a modification to the original code, we have added an adhoc fix that exits the line search in the case of

\[
\Phi(\alpha_l) - \Phi(0) > 0.
\]

This has been added as experience have shown that \textsc{vmcon} typically does not converge in these situations, but if it is forced to calculate a new search direction in this way, it sometimes successfully finishes. Note, that this cannot force \textsc{vmcon} to converge on any false solutions, as it only exits positively when the convergence criterion \((A.5)\) is fulfilled.

Typically, the line search already converges after one iteration and, therefore, \(\alpha = 1\). Hence, the \textsc{vmcon} line search has an upper limit of maximally 10 iterations before it terminating with \(ifail = 3\) (c.f. Table \((A.1)\). This is higher than Powell’s original limit of 5 to avoid a few cases of early termination without a major effect on efficiency.

Within the line search \textsc{vmcon} also checks that the total number of function calls has not exceeded \(maxfev = 100\). If this is the case, it exits with error code \(ifail = 2\). Both checks assure that the routine stops, if it does not seem to be able to find a solution.

\section{A.7 The Broyden-Fletcher-Goldfarb-Shanno (BFGS) quasi-Newton update} \label{sec:a.7}

\textsc{vmcon} uses a quasi-Newtonian update, the Broyden-Fletcher-Goldfarb-Shanno update, in the approximation of the Hessian of the Lagrange function. This means it is applicable to all continuously differentiable objective and constraint functions. Note, that due to the constraints it is not actually necessary for the Hessian of the solution to be positive definite, even though this is essential for the convergence of the algorithm.

In quasi-Newtonian methods the identity matrix \(I\) is often chosen as an initial estimate of the Hessian because of its positive definiteness. In some cases it can be helpful to chose a constant multiple of the

\footnote{In the actual code \(\alpha = \text{calpha} \) and \(\alpha_l = \text{alpha} \times \alpha_{l-1}\).}
identity matrix instead. The BGFS update \([7]\) is one way of revising the initial Hessian approximation using the update of the iteration variable vector

\[
\xi = x^j - x^{j-1}
\]  
(A.15)

and the update of the Jacobian of the Lagrange function

\[
\gamma = \nabla_x L(x^j, \lambda^j) - \nabla_x L(x^{j-1}, \lambda^j).
\]  
(A.16)

Note that unless \(\alpha = 1\) in the line search, \(\xi \neq \delta\). To assure the positive definiteness of \(B\) Powell \([6]\) suggested a variation of the standard formalism that uses \(\eta\) instead of \(\gamma\)

\[
\eta = \theta \gamma + (1 - \theta)B\xi
\]  
(A.17)

with

\[
\theta = \begin{cases} 
1 & \xi^T \gamma \geq 0.2\xi^T B\xi \\
0.8\xi^T B\xi & \xi^T \gamma < 0.2\xi^T B\xi 
\end{cases}
\]  
(A.18)

to calculate a new approximation of the Hessian

\[
B_{\text{new}} = B - \frac{B\xi\xi^TB + \eta^T\eta}{\xi^TB\xi + \xi^T\eta},
\]  
(A.19)

Using the modified version of the BFGS update as suggested by Powell, furthermore, assures superlinear convergence, even if the true Hessian is indefinite \([82]\).

A.8 Symbols

In the previous sections the following conventions have been assumed: \(x, \delta, \xi, \gamma, \text{ and } \eta\) are \(n\)-dimensional vectors, where \(n\) is the number of iteration variables. \(c, \lambda, \mu\) are \(m\)-dimensional vectors, where \(m\) is the total number of constraints and \(c_i, \lambda_i, \text{ and } \mu_i\) (\(i = 1 \ldots m\)) are their respective components. \(B\) and \(I\) are \(n \times n\)-dimensional matrices, while \(\nabla_x c\) is an \(n \times m\)-dimensional matrix.
Appendix B

The Input File

The input file IN.DAT is used to change the values of the physics, engineering and other code parameters from their default values, and to set up the numerics (constraint equations, iteration variables etc.) required to define the problem to be solved. The user interface writes the input file, so it is not necessary to edit it directly.

B.0.1 Tokamak, stellarator, RFP or IFE?

The default model is the tokamak. To select a stellarator, reversed field pinch or inertial fusion energy plant, an additional input file is required, device.dat, which should contain a single character in the first line, which is interpreted as follows:

0 : use tokamak model
1 : use stellarator model
2 : use reversed field pinch model
3 : use inertial fusion energy model

B.0.2 File format

Variables can be specified in any order in the input file. Comment lines start with a * character. Data lines are of the form

variable = value

where variable is the name of one of the input parameters or iteration variables listed in the variable descriptor file, and value is the (usually numerical) initial value required for that variable. (Arrays, as opposed to scalar quantities, are treated differently — see below.) All input data are screened for non-sensible values.

The following rules must be obeyed when writing an input file:

1. Each variable must be on a separate line.
2. Variable names can be upper case, lower case, or a mixture of both.
3. Spaces may not appear within a variable name or data value.
4. Other spaces within a line, and trailing spaces, are ignored.

5. Commas are not necessary between variables (but see below).

6. Data can extend over more than one line.

7. One-dimensional arrays can be explicitly subscripted, or unscripted, in which case the following element order is assumed: \( A(1), A(2), A(3), \ldots \)

8. At present, multiple dimension arrays can only be handled without reference to explicit subscripts, in which case the following element order is assumed: \( B(1,1), B(2,1), B(3,1), \ldots \)

   The use of the input file to specify multiple dimension array elements is prone to error.

9. Unscripted array elements must be separated by commas.

10. Blank lines are allowed anywhere in the input file.

11. Lines starting with a `*` are assumed to be comments.

12. Comment lines starting with five or more asterisks (i.e. `*****`) are reproduced verbatim in the output file. This feature is not recommended, as these comments are likely to become out of date. The user interface does not support this feature.

13. In-line comments are *usually* ignored, but there can be problems if one contains a comma (,). If this is the case, there must also be a comma after the variable’s value and before the comment.

   It is useful to divide the input file into sections, using suitable comment lines, to help the user keep related variables together.

   The following is a valid fragment of an input file (the vertical lines are simply to help show the column alignment):

   ```
   * This line is a comment that will not appear in the output
   ***** This line is a comment that will appear in the output
   bound(1) = 2.5,
   BOUNDU(10) = 3. ,
   BOUNDU(45) = 1 ,
   * Another comment... Note that real values can be entered as if
   * they were integers, and vice versa (but it’s not recommended...)
   epsfcn = 10.e-4,
   Ftol = 1.D-4,
   * The next line sets the first five elements of array icc:
  ICC = 2, 10, 11, 24, 31
   * The next line sets the first ten elements of array ixc:
   ixc = 10, 12, 3, 36, 48,
   1, 2, 6, 13, 16,
   IOPTIMZ = 1,
   maxcal = 200
   nsweep = 7
   NEQNS = 5, This is an in-line comment
   NVAR = 10, Another, but successfully containing a comma!
   ```

   The following are *invalid* entries in the input file (Q: Why?!):
Appendix B  The Input File

| boundl(1,1) = 2.5, |
| BOUNDU(N) = 3.0, |
| A line of ‘random’ characters like this will clearly wreak havoc |
| eps fcn = 10.e-4, ftol = 1.D-4 |
| epsvmc = 1.0 e-4 |
| ICC = 2 10 11 24 31 |
| IOPTIMZ = 1.0, This will in fact be okay - but is not recommended |
| NEQNS = 5 An in-line comment on a line with only one comma (,) character |

If the code encounters a problem reading the input file, it will stop immediately with a (hopefully) useful error message. It may be worth looking at the contents of the output file as well, to help narrow down on which line of the input file the problem might lie.
Appendix C

Optimisation Input File

The following is a typical input file used to run PROCESS in optimisation mode.

*****
runtitle = 'Example input file'

*--------------------------------------------------*

*---------------Constraint Equations---------------*
neqns = 18
icc(1) = 1 * beta (consistency equation)
icc(2) = 2 * global power balance (consistency equation)
icc(3) = 5 * density upper limit
icc(4) = 8 * neutron wall load upper limit
icc(5) = 10 * toroidal field 1/r (consistency equation)
icc(6) = 11 * radial build (consistency equation)
icc(7) = 13 * burn time lower limit (pulse)
icc(8) = 15 * l-h power threshold limit
icc(9) = 16 * net electric power lower limit
icc(10) = 24 * beta upper limit
icc(11) = 26 * central solenoid eof current density upper limit
icc(12) = 27 * central solenoid bop current density upper limit
icc(13) = 30 * injection power upper limit
icc(14) = 31 * tf coil case stress upper limit (sctf)
icc(15) = 32 * tf coil conduit stress upper limit (sctf)
icc(16) = 33 * i_op / i_critical (tf coil) (sctf)
icc(17) = 34 * dump voltage upper limit (sctf)
icc(18) = 35 * j_winding pack/j_protection upper limit (sctf)

*---------------Iteration Variables----------------*
nvar = 31
ixc(1) = 2 * bt * Toroidal field on axis (t) (iteration variable 2)
boundl(2) = 0.01
boundu(2) = 100.0

ixc(2) = 3 * rmajor * Plasma major radius (m) (iteration variable 3)
boundl(3) = 0.1
boundu(3) = 13.0

ixc(3) = 4 * te * Volume averaged electron temperature (kev)
boundl(4) = 5.0
boundu(4) = 150.0

ixc(4) = 5 * beta * Total plasma beta (iteration variable 5)
boundl(5) = 0.001
boundu(5) = 1.0
ixc(5) = 6  * dene * Electron density (/m3) (iteration variable 6)
boundl(6) = 1e+19
boundu(6) = 1e+21

ixc(6) = 9  * fdene * F-value for density limit
boundl(9) = 0.001
boundu(9) = 1.2

ixc(7) = 10  * hfact * H factor on energy confinement times (iteration variable 10)
boundl(10) = 0.1
boundu(10) = 1.1

ixc(8) = 12  * oacdp * Overall current density in tf coil inboard legs (a/m2)
boundl(12) = 100000.0
boundu(12) = 150000000.0

ixc(9) = 13  * tfcth * Inboard tf coil thickness; (centrepost for st) (m)
boundl(13) = 0.5
boundu(13) = 5.0

ixc(10) = 14  * fwalld * F-value for minimum wall load
boundl(14) = 0.001
boundu(14) = 1.0

ixc(11) = 16  * ohcth * Central solenoid thickness (m)
boundl(16) = 0.001
boundu(16) = 100.0

ixc(12) = 18  * q * Safety factor 'near' plasma edge (iteration variable 18)
boundl(18) = 3.0
boundu(18) = 100.0

ixc(13) = 29  * bore * Central solenoid inboard radius (m)
boundl(29) = 0.1
boundu(29) = 10.0

ixc(14) = 36  * fbetatry * F-value for beta limit
boundl(36) = 0.001
boundu(36) = 1.0

ixc(15) = 37  * coheof * Central solenoid overall current density at end of flat-top (a/m2)
boundl(37) = 100000.0
boundu(37) = 100000000.0

ixc(16) = 38  * fjohc * F-value for central solenoid current at end-of-flattop
boundl(38) = 0.01
boundu(38) = 0.25

ixc(17) = 39  * fjohc0 * F-value for central solenoid current at beginning of pulse
boundl(39) = 0.001
boundu(39) = 0.25

ixc(18) = 41  * fcohbop * Ratio of central solenoid overall current density at
boundl(41) = 0.001
boundu(41) = 1.0

ixc(19) = 42  * gapoh * Gap between central solenoid and tf coil
boundl(42) = 0.05
boundu(42) = 0.1

ixc(20) = 44  * fvsbrnii * Fraction of the plasma current produced by
Appendix C  Optimisation Input File

boundl(44) = 0.001
boundu(44) = 1.0

ixc(21) = 48  * fstrcase * F-value for tf coil case stress
boundl(48) = 0.001
boundu(48) = 1.0

ixc(22) = 49  * fstrcond * F-value for tf coil conduit stress
boundl(49) = 0.001
boundu(49) = 1.0

ixc(23) = 50  * fiooic * F-value for tf coil operating current / critical
boundl(50) = 0.001
boundu(50) = 0.5

ixc(24) = 51  * fvdump * F-value for dump voltage
boundl(51) = 0.001
boundu(51) = 1.0

ixc(25) = 53  * fjprot * F-value for tf coil winding pack current density
boundl(53) = 0.001
boundu(53) = 1.0

ixc(26) = 56  * tdmptf * Dump time for tf coil (s)
boundl(56) = 10.0
boundu(56) = 1000000.0

ixc(27) = 57  * thkcas * Inboard tf coil case outer (non-plasma side) thickness (m)
boundl(57) = 0.05
boundu(57) = 1.0

ixc(28) = 58  * thwcndut * Tf coil conduit case thickness (m)
boundl(58) = 0.004
boundu(58) = 1.0

ixc(29) = 59  * fcutfsu * Copper fraction of cable conductor (tf coils)
boundl(59) = 0.001
boundu(59) = 1.0

ixc(30) = 61  * gapds * Gap between inboard vacuum vessel and tf coil (m)
boundl(61) = 0.12
boundu(61) = 10.0

ixc(31) = 103 * flhthresh * F-value for l-h power threshold
boundl(103) = 1.0
boundu(103) = 1000000.0

*-----------------Build Variables------------------*
blnkith = 0.755  * Inboard blanket thickness (m);
blnkoth = 1.275  * Outboard blanket thickness (m);
bore = 2.5907   * Central solenoid inboard radius (m)
ddwx = 0.15     * External cryostat thickness (m)
ddwi = 0.32     * Vacuum vessel thickness (tf coil / shield) (m)
fvith = 0.025    * Inboard first wall thickness; initial estimate (m)
fvoth = 0.025   * Outboard first wall thickness; initial estimate (m)
gapds = 0.12    * Gap between inboard vacuum vessel and tf coil (m)
gapoh = 0.05    * Gap between central solenoid and tf coil
gapomin = 0.2   * Minimum gap between outboard vacuum vessel and tf coil (m)
ohcth = 0.86365 * Central solenoid thickness (m)
scrapli = 0.225  * Gap between plasma and first wall; inboard side (m)
scraplo = 0.225  * Gap between plasma and first wall; outboard side (m)
shldith = 0.3   * Inboard shield thickness (m)
shldoth = 0.8  * Outboard shield thickness (m)
shldtth = 0.3  * Upper/lower shield thickness (m);
tfcth = 0.79433 * Inboard tf coil thickness; (centrepost for st) (m)
vgap2 = 0.12   * Vertical gap between vacuum vessel and tf coil (m)
vgaprf = 1.6   * Vertical gap between x-point and divertor (m)

*---------------Buildings Variables----------------*

*---------------Constraint Variables---------------*
bmxlim = 14.0  * Maximum peak toroidal field (t)
fbetatry = 0.52645 * F-value for beta limit
fdene = 1.2   * F-value for density limit
fhldiv = 2.0  * F-value for divertor heat load
fjohc = 0.25  * F-value for central solenoid current at end-of-flattop
fjohc0 = 0.25 * F-value for central solenoid current at beginning of pulse
flhthresh = 1.0232 * F-value for l-h power threshold
fpeakb = 0.9229 * F-value for maximum toroidal field
fstrcond = 0.84986 * F-value for tf coil conduit stress
fvdump = 0.96778 * F-value for dump voltage
fwalld = 0.13514 * F-value for minimum wall load
pnetelin = 500.0 * Required net electric power (mw)
pseprmax = 17.0 * Maximum ratio of power crossing the separatrix to
pnetelin = 500.0 * Required net electric power (mw)
tbrnmn = 7200.0 * Minimum burn time (s)
walalw = 8.0   * Allowable wall-load (mw/m2)

*------------------Cost Variables------------------*
abktflnc = 15.0  * Allowable first wall/blanket neutron
adivflnc = 20.0  * Allowable divertor heat fluence (mw-yr/m2)
fcap0 = 1.15  * Average cost of money for construction of plant
fcap0cp = 1.06  * Average cost of money for replaceable components
fcontng = 0.15  * Project contingency factor
fcr0 = 0.065  * Fixed charge rate during construction
ifueltyp = 1  * Switch * treat blanket divertor; first wall and
lsa = 2  * Level of safety assurance switch
output_costs = 0  * Switch for costs output * do not write cost-related outputs to file;
ratecdol = 0.06  * Effective cost of money in constant dollars
tlif = 40.0  * Plant life (years)
uchld = 280.0  * Unit cost for blanket vanadium ($/kg)
ucdiv = 500000.0  * Cost of divertor blade ($) 
ucme = 30000000.0  * Unit cost of maintenance equipment ($/w**0.3)

*------------------Current Drive Variables------------------*
bscfmax = 0.99  * Maximum fraction of plasma current from bootstrap;
etanbi = 0.4  * Neutral beam wall plug to injector efficiency
frbeam = 1.0  * R_tangential / r_major for neutral beam injection
pinjalw = 50.0  * Maximum allowable value for injected power (mw)

*------------------Divertor Variables------------------*
anginc = 0.175  * Angle of incidence of field line on plate (rad)
divdum = 1  * Switch for divertor zeff model* 0=calc; 1=input
divfix = 0.621  * Divertor structure vertical thickness (m)
hldivlim = 10.0  * Heat load limit (mw/m2)
ksic = 1.4  * Power fraction for outboard double-null scrape-off plasma
prn1 = 0.4  * N-scrape-off / n-average plasma;
zeffdiv = 3.5  * Zeff in the divertor region (if divdum /= 0)
Appendix C  Optimisation Input File

*------------------Fwbs Variables------------------*
fwclfr = 0.1  * First wall coolant fraction

*-----------------Global Variables-----------------*

*-----------------Heat Transport Variables-----------------*
etath = 0.375  * Thermal to electric conversion efficiency
fauxbop = 0.032  * Fraction of gross electric power to balance-of-plant
ffwlg = 0.01  * Fraction of first wall / divertor power to low grade heat
htpmw = 155.0  * Heat transport system electrical pump power (mw)

*------------------Ife Variables-------------------*

*------------Impurity Radiation Module-------------*
coreradius = 0.6  * Normalised radius defining the 'core' region
fimp = 1.0, 0.1, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.00044, 5e-05  * Impurity number density fractions
impvar = 13  * Fimp element value to be varied if iteration

*---------------------Numerics---------------------*
ioptimz = 1  * Code operation switch * for no optimisation; hybrd only;
minmax = 1  * Switch for figure-of-merit * major radius
epsvmc = 1e-08  * Error tolerance for vmcon

*----------------Pf Power Variables----------------*

*-----------------Pfcoil Variables-----------------*
coheof = 12994000.0  * Central solenoid overall current density at end of flat-top (a/m2)
cptdin = 42200.0, 42200.0, 42200.0, 42200.0, 43000.0, 43000.0, 43000.0, 43000.0, 43000.0  * Peak current per turn input for
fchbop = 0.93314  * Ratio of central solenoid overall current density at
ipfloc = 2, 2, 3, 3  * Switch for locating scheme of pf coil group i
isumatpf = 3  * Switch for superconductor material in pf coils * nbti;
ncls = 1, 1, 2, 2  * Number of pf coils in group j
ngrp = 4  * Number of groups of pf coils;
ohhghf = 0.9  * Central solenoid height / tf coil internal height
rjconpf = 11000000.0, 11000000.0, 6000000.0, 6000000.0, 8000000.0, 8000000.0, 8000000.0, 8000000.0, 8000000.0  * Average winding
rpf2 = -1.825  * Offset (m) of radial position of ipfloc=2 pf coils
sigpfalw = 300.0  * Allowable stress in pf coils/central solenoid (mpa)
zref = 3.6, 1.2, 1.0, 2.8, 1.0, 1.0, 1.0, 1.0  * Pf coil vertical positioning adjuster

*-----------------Physics Variables-----------------*
alphaj = 2.0  * Current profile index;
alphan = 1.0  * Density profile index
alphat = 1.0  * Temperature profile index
aspect = 3.5  * Aspect ratio (iteration variable 1)
beta = 0.034253  * Total plasma beta (iteration variable 5)
bt = 5.3983  * Toroidal field on axis (t) (iteration variable 2)
dene = 8.2808e+19  * Electron density (/m^3) (iteration variable 6)
dnbeta = 3.0  * (troyon-like) coefficient for beta scaling;
fkzohm = 1.0245  * Zohm elongation scaling adjustment factor (ishape=2; 3)
fvsbrnni = 0.43898  * Fraction of the plasma current produced by
gamma = 0.3  * Ejima coefficient for resistive startup v-s formula
hfact = 1.1  * H factor on energy confinement times (iteration variable 10)
ibss = 4  * Switch for bootstrap current scaling * for sauter et al scaling
iculbl = 1  * Switch for beta limit scaling * apply limit to thermal beta;
impc = 0.0  * Carbon impurity multiplier (imprad_model=0 only)
imo = 0.0  * Oxygen impurity multiplier (imprad_model=0 only)
nped = 6.78e+19 * Electron density of pedestal (/m3) (ipedestal=1)
nsep = 2e+19  * Electron density at separatrix (/m3) (ipedestal=1)
rhopedn = 0.94  * R/a of density pedestal (ipedestal=1)
rhopedt = 0.94  * R/a of temperature pedestal (ipedestal=1)
tped = 5.5  * Electron temperature of pedestal (kev) (ipedestal=1)
tsep = 0.1  * Electron temperature at separatrix (kev) (ipedestal=1)
ishape = 2  * Switch for plasma cross-sectional shape calculation * set kappa to the natural elongation value
kappa = 1.7  * Plasma separatrix elongation (calculated if ishape > 0)
rmajor = 8.9216  * Plasma major radius (m) (iteration variable 3)
te = 12.959  * Volume averaged electron temperature (kev)
triang = 0.5  * Plasma separatrix triangularity (calculated if ishape=1; 3 or 4)
zfear = 1  * High-z impurity switch; 0=iron; 1=argon

*-----------------Pulse Variables------------------*
lpulse = 1  * Switch for reactor model * pulsed operation

*------------------Rfp Variables-------------------*

*-------------------Scan Module--------------------*
isweep = 1  * Number of scan points to calculate
nsweep = 1  * Switch denoting quantity to scan * aspect
sweep = 3.1  * Actual values to use in scan

*--------------Stellarator Variables---------------*

*-----------------Tfcoil Variables------------------*
casths = 0.05  * Inboard tf coil sidewall case thickness (m)
cpttf = 65000.0  * Tf coil current per turn (a)
csytf = 990000000.0  * Yield strength of case (tf coils and cs coils) (pa)
cutfsu = 0.61471  * Copper fraction of cable conductor (tf coils)
oacdp = 12367000.0  * Overall current density in tf coil inboard legs (a/m2)
ripmax = 0.6  * Maximum allowable toroidal field ripple amplitude
tfno = 18.0  * Number of tf coils (default = 50 for stellarators)
tftmp = 4.75  * Peak helium coolant temperature in tf coils and pf coils (k)
thicndut = 0.001  * Conduit insulation thickness (m)
thkcas = 0.43968  * Inboard tf coil case outer (non-plasma side) thickness (m)
thwcondt = 0.004  * Tf coil conduit case thickness (m)
tmargmin = 1.7  * Minimum allowable temperature margin (cs and tf coils) (k)
vftf = 0.333  * Coolant fraction of tf coil leg (itfssup=0)

*-----------------Times Variables------------------*
tburn = 10000.0  * Burn time (s) (calculated if lpulse=1)

*-----------------Vacuum Variables------------------*
Appendix D

Non-optimisation Input File

The following is a typical input file used to run PROCESS in non-optimisation mode. Comments in [...] have been added to the right of each line.

* Numerics information

NEQNS = 14, [Number of active constraint equations]
NVAR = 14, [Number of active iteration variables]
ICC = 1, 2, 10, 11, 16, 24, 5, 31, 32, 33, 34, 35, 36, [Constraint eqns]
ixc = 5, 10, 12, 3, 7, 6, 36, 9, 48, 49, 50, 51, 53, 54, [Iteration variables]
IOPTMNZ = -1, [Turn off optimisation]
ISWEEP=0 [No scans (non-optimisation mode)]

* F-values and limits

FBETATRY = 1.0 [N.B. active iteration variable 36]

* Physics parameters

ASPECT = 3.5, [Machine aspect ratio]
BETA = 0.042, [N.B. active iteration variable 5]
BT = 6., [Toroidal field on axis]
DENCI = 1.5e20, [N.B. active iteration variable 6]
FVSBRNNI = 1.0, [Non-inductive volt seconds fraction]
DNBETA = 3.5, [Beta g coefficient]
HFACT = 2., [N.B. active iteration variable 10]
ICURR = 4, [Use ITER current scaling]
ISC = 6, [Use ITER 89-P confinement time scaling law]
IINVQD = 1, [Use inverse quadrature]
IITER = 1, [Use ITER fusion power calculations]
ISHAPE = 0, [Use input values for KAPPA and TRIANG]
KAPPA = 2.218, [Plasma elongation]
Q = 3.0, [Edge safety factor]
RMAJOR = 7.0, [N.B. active iteration variable 3]
RNBEAM = 0.0002, [N.B. active iteration variable 7]
TBURN = 227.9 [Burn time]
TE = 15., [Electron temperature]
TRIANG = 0.6 [Plasma triangularity]

* Current drive parameters

IRFCD = 1, [Use current drive]
IEFRF = 5 [Use ITER neutral beam current drive]
FEFFCD = 3., [Artificially enhance efficiency]
* Divertor parameters

\begin{align*}
\text{ANGINC} &= 0.262, \quad \text{[Angle of incidence of field lines on plate]} \\
\text{PRN1} &= 0.285 \quad \text{[Density ratio]}
\end{align*}

* Machine build

\begin{align*}
\text{BORE} &= 0.12, \quad \text{[Machine bore]} \\
\text{OHCTH} &= 0.1, \quad \text{[central solenoid thickness]} \\
\text{GAPOH} &= 0.08, \quad \text{[Inboard gap]} \\
\text{TFCTH} &= 0.9, \quad \text{[Inboard TF coil leg thickness]} \\
\text{DDWI} &= 0.07, \quad \text{[Vacuum vessel thickness]} \\
\text{SHLDITH} &= 0.69, \quad \text{[Inboard shield thickness]} \\
\text{BLNKITH} &= 0.115, \quad \text{[Inboard blanket thickness]} \\
\text{FWITH} &= 0.035, \quad \text{[Inboard first wall thickness]} \\
\text{SCRAPLI} &= 0.14, \quad \text{[Inboard scrape-off layer thickness]} \\
\text{SCRAPLO} &= 0.15, \quad \text{[Outboard scrape-off layer thickness]} \\
\text{FWOTH} &= 0.035, \quad \text{[Outboard first wall thickness]} \\
\text{BLNKOTH} &= 0.235, \quad \text{[Outboard blanket thickness]} \\
\text{SHLDOTH} &= 1.05, \quad \text{[Outboard shield thickness]} \\
\text{GAPOMIN} &= 0.21, \quad \text{[Outboard gap]} \\
\text{VGAPTF} &= 0, \quad \text{[Vertical gap]}
\end{align*}

* First wall, blanket, shield parameters

\begin{align*}
\text{LBLNKT} &= 0 \quad \text{[Use old blanket model]} \\
\text{DENSTL} &= 7800. \quad \text{[Steel density]}
\end{align*}

* TF coil parameters

\begin{align*}
\text{OACDCP} &= 1.467, \quad \text{[N.B. active iteration variable 12]} \\
\text{ITFSUP} &= 1, \quad \text{[Use superconducting TF coils]} \\
\text{RIPMAX} &= 5. \quad \text{[Maximum TF ripple]}
\end{align*}

* PF coil parameters

\begin{align*}
\text{NGRP} &= 3, \quad \text{[Three groups of PF coils]} \\
\text{IPFLOC} &= 1,2,3, \quad \text{[Locations for each group]} \\
\text{NCLS} &= 2,2,2,1, \quad \text{[Number of coils in each group]} \\
\text{COHEOP} &= 1.85e7, \quad \text{[central solenoid current at End Of Flat-top]} \\
\text{FCOHBO} &= 0.9, \quad \text{[central solenoid current at Begin. Of Pulse / COHEOF]} \\
\text{ROUTR} &= 1.5, \quad \text{[Radial position for group 3]} \\
\text{ZREF(3)} &= 2.5, \quad \text{[Z position for group 3]} \\
\text{OHHGHF} &= .71 \quad \text{[Height ratio central solenoid / TF coil]}
\end{align*}

* Vacuum system parameters

\begin{align*}
\text{NTYPE} &= 1 \quad \text{[Use cryopump]}
\end{align*}

* Heat transport parameters

\begin{align*}
\text{ETATH} &= 0.35 \quad \text{[Thermal to electric conversion efficiency]} \\
\text{FMGDMW} &= 0. \quad \text{[Power to MGF units]} \\
\text{BASEEL} &= 5.66 \quad \text{[Base plant electric load]} \\
\text{ISCEEN} &= 2 \quad \text{[Energy store option]}
\end{align*}

* Buildings

\begin{align*}
\text{FNDT} &= 2. \quad \text{[Foundation thickness]} \\
\text{EFLOOR} &= 1.45 \quad \text{[Effective total floor space]}
\end{align*}
* Costs

IREACTOR = 1, [Calculate cost of electricity]
IFUELTY = 0 [Treat blanket, first wall etc as capital cost]
UCHRS = 87.9, [Unit cost of heat rejection system]
UCCPCL1 = 250, [Unit cost of high strength tapered copper]
UCCPCLB = 150 [Unit cost of TF outer leg plate coils]