ANSTO/E726

ANSTO/E726

ansto

POW3D - NEUTRON DIFFUSION MODULE OF THE AUS SYSTEM, A USER'S MANUAL

by

B. V. HARRINGTON

J. P. POLLARD

J. M. BARRY

November 1996

ISBN 0 642 59964 5

ISSN 1030-7745

REPORT E

AUSTRALIAN NUCLEAR SCIENCE
AND TECHNOLOGY ORGANISATION

LUCAS HEIGHTS SCIENCE AND TECHNOLOGY CENTRE

POW3D - NEUTRON DIFFUSION MODULE OF THE AUS SYSTEM, A USER'S MANUAL

by

B. V. Harrington, J. P. Pollard and J. M. Barry

ABSTRACT

POW3D is a three-dimensional neutron diffusion module of the AUS modular neutronics code system. It performs eigenvalue, source or feedback-free kinetics calculations. The module includes general criticality search options and extensive editing facilities including perturbation calculations. Output options include flux or reaction rate plot files. The code permits selection from one of a variety of different solution methods (MINI, ICCG or SLOR) for inner iterations with region rebalance to enhance convergence. A MINI-accelerated Gauss-Siedel method is used for upscatter iterations with group rebalance to enhance convergence. Chebyshev source extrapolation is applied for outer iterations.

ISBN 0 642 59964 5
ISSN 1030-7745

The following descriptors have been selected from the INIS Thesaurus to describe the subject matter of this report for information retrieval purposes. For further details please refer to IAEA-INIS-12 (INIS: Manual for Indexing) and IAEA-INIS-13 (INIS: Thesaurus) published in Vienna by the International Atomic Energy Agency.

COMPUTER PROGRAM DOCUMENTATION; COMPUTER CALCULATIONS; CONVERGENCE; CRITICALITY; CROSS SECTIONS; EIGENVALUES; ITERATIVE METHOD; MANY-DIMENSIONAL CALCULATIONS; MOATA REACTOR; MULTIGROUP THEORY; NEUTRON DIFFUSION EQUATION; NEUTRON FLUX; NEUTRON LEAKAGE; P CODES; PERTURBATION THEORY; REACTOR KINETICS

EDITORIAL NOTE

The Australian Nuclear Science and Technology Organisation (ANSTO) replaced the Australian Atomic Energy Commission (AAEC) on 27 April 1987. Reports issued after April 1987 have the prefix ANSTO with no change of the symbol (E, M, S or C) or the numbering sequence.

CONTENTS

1. INTRODUCTION

A versatile, fast and accurate multidimensional, multigroup neutron diffusion code is vital to any reactor neutronics calculation scheme. In the early days on our ANSTO IBM360/50 computer we mainly used the 2D codes CRAM (Hassitt 1962, rewritten in FORTRAN originally for the IBM7040) and GOG (Hopkins and Oakes 1968, somewhat rewritten for the IBM360/50). A locally developed code, POW [Pollard 1974] replaced these codes in the mid 1970s. In the 1980s a replacement for POW appeared, POW3D, which was capable of three-dimensional calculations. Even so we did not then have the hardware to exploit the extra dimension for everyday calculations and POW continued to be used. With the increased capability of the Fujitsu VP2200 Supercomputer at ANSTO, three-dimensional calculations are feasible and POW was replaced by POW3D in 1987.

Like its predecessor, POW3D is a general purpose, multigroup neutron diffusion code and runs on the computer as part of the AUS modular neutronics code scheme [Robinson 1987 and 1991]. Both POW and POW3D rely on an edge mesh representation of flux for accurate estimation of leakage throughout a reactor [Wachspress 1966]. POW3D can also be used for multidimensional kinetic studies for systems perturbed from the steady state with a prescribed pulsed (feedback-free) variation of some, usually physical, parameter; for example the concentration of a reactor material.

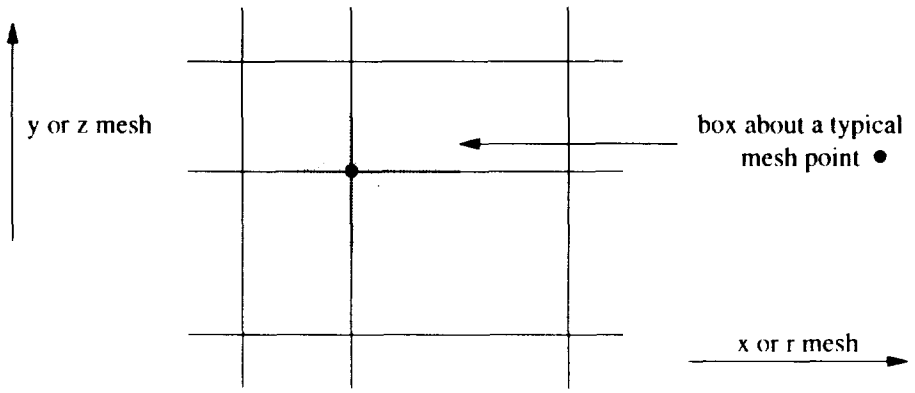
The main difference between POW and POW3D, beside the x,y,z geometry option, is the variety of solution techniques available. The Method of Implicit Non-stationary Iteration (MINI) was developed for solving large sets of linear equations such as arise in diffusion studies [Barry and Pollard 1977, 1979 and Barry 1982]. The MINI procedure hastens convergence compared with other iterative procedures particularly for 3D calculations and when extensive upscatter is present. For this reason MINI is the default procedure although others discussed in Appendix D may easily be selected.

As expected, POW3D input data is an extension of POW data. Both codes rely on free format input for ease of use. POW3D uses the keyword controlled free input routine SKAN [Pollard 1978]. Comprehensive editing options include production of flux or reaction rate plot files and perturbation analysis. Mathematical details of POW3D are presented by Barry and Pollard [1987 revised 1995].

The source language of POW3D is mainly FORTRAN 77 for a UNIX machine. POW3D was originally designed to run on IBM machines under MVS but now runs on a Fujitsu VP2200 Supercomputer and a Silicon Graphics workstation under UNIX. The program adjusts itself to whatever random access memory is made available but normally requires a minimum of 3 Megabytes although on the Fujitsu VP2200 30 Megabytes are usually used. The code makes heavy use of slower storage in the form of disks when sufficient memory is not available, which greatly extends the time for a job to finish.

2. THE MULTIGROUP NEUTRON DIFFUSION EQUATION

The general form of the multigroup neutron diffusion equations, solved by POW3D at each mesh point for the flux in a box about each mesh point, is obtained from a detailed analysis of neutron events [Stacey 1969]. The method used to solve the equation is a finite difference method generating a numerical solution at the intersection of selected mesh as illustrated by a 2D example.



We therefore cast the neutron diffusion equations into the form required, namely

$$-\frac{1}{v_g}\int_{box}\frac{\partial\phi_g}{\partial t}\,dV = -\int_{box}\nabla\cdot D_{n,g}\,\nabla\phi_g\,dV + \phi_g\int_{box}\sigma_{rem,g}\,dV -$$
$$-\sum_{g'}\phi_{g'}\int_{box}\sigma_{s,g'\to g}\,dV - \chi_{p,g}\,(1-\beta)\sum_{g'}\phi_{g'}\int_{box}\frac{\upsilon}{k}\,\sigma_{f,g'}\,dV -$$
$$-\sum_{i}\chi_{i,g}\,\lambda_i\,C_i - S_g\,, \qquad \text{for } g = 1, 2, \cdots ng.$$

In addition we have the precursor concentration equations for igd delayed neutron groups

$$\frac{\partial C_i}{\partial t} = \beta_i\sum_{g'}\phi_{g'}\int_{box}\frac{\upsilon}{k}\,\sigma_{f,g'}\,dV - \lambda_i C_i\,, \qquad \text{for } i = 1, 2, \cdots, igd,$$

where

k	is the effective steady state multiplication constant,
$S_g(\underline{r}, t)$	is the external source in each box for energy group g,
$\chi_{p,g}$	is the prompt fission spectrum (normalised to unit sum),
$\chi_{i,g}$	is the delayed fission spectrum (normalised to unit sum), for delayed group i yielding fraction β_i of total emissions, $\beta(=\sum_i \beta_i)$ of which are delayed,
$\sigma(\underline{r}, t)$	denotes various (macroscopic) cross-sections,
$\sigma_{s,g'\to g}(\underline{r}, t)$	is the scattering matrix for transfers from group g′ to g (and the self scatter term, $\sigma_{s,g\to g}$, is taken as zero),
$D_{n,g}(\underline{r}, t)$	denotes possibly directional diffusion coefficients (for direction n or -n parallel to the chosen axes),
v_g	denotes average velocity in group g,
$C_i(\underline{r}, t)$	is the total concentration in each box of the precursor,
λ_i	is the decay constant of delayed group i,
$\phi_g(\underline{r}, t)$	is the group g flux in each box required to be calculated,
$\underline{r}$	denotes the coordinates of a typical mesh point (or a point in the box),
t	is time

and sums are taken over all neutron energy groups (1, 2,...,ng) and all delayed neutron groups (1, 2,...,igd). The equations are solved for ϕ_g subject to certain conditions.

(i) **the outer boundary conditions -**

(a) **reflective** (zero current)

$$\frac{\partial\phi_g}{\partial n} = 0\,, \quad \text{for each group g,}$$

where n = outward normal,

or

(b) **reactor** (zero flux on extrapolated boundary)

$$D_{n,g}\frac{\partial\phi_g}{\partial n} + \frac{\phi_g}{3d} = 0\,, \quad \text{for each group g,}$$

where d = extrapolation distance in transport mean free paths (usually 0.71 is used);

(ii) **the internal boundary conditions** for boundaries separating different materials say L and R -

(a) **continuity of flux**

$$\phi_g|_L - \phi_g|_R = 0, \quad \text{for each group g,}$$

and

(b) **continuity of current**

$$n \cdot (D_{n,g} \nabla\phi_g|_L - D_{n,g} \nabla\phi_g|_R) = 0, \quad \text{for each group g,}$$

where n = boundary normal;

(iii) **the initial conditions for kinetics** -

(a) **steady state**

$$\frac{\partial\phi_g}{\partial t} = 0, \quad \frac{\partial C_i}{\partial t} = 0, \quad S_g = 0, \quad \text{for } t \le 0 \text{ and for all groups,}$$

$$\sum_{g'} \int_{\text{reactor}} f\sigma_{f,g'} \phi_{g'} \, dV|_{t=0} = P,$$

where f = fission energy release,
P = specified power,

which requires POW3D to solve an eigenvalue problem for either k or for λ (used to multiply a physical parameter to give criticality); for a kinetics study the steady state value of k must be retained in the equations so that any time dependent variation is attributable to a variation from the steady state,

or

(b) **shut down**

$$\phi_g = 0, \; C_i = 0, \; S_g = 0, \quad \text{for } t \le 0 \text{ and for all groups.}$$

3. NUMERICAL METHODS - OVERVIEW

In this section we outline the numerical methods used in POW3D. Some details for the spatial solution options are given in Appendix D.

The central consideration for time dependence over a small time step from t_{p-1} to t_p, is that we may approximate integrals of the general type

$$\int_{t_{p-1}}^{t_p} f(t) \, \phi_g(\underline{r}, t) \left(\int_{\text{box}} \sigma_g(\underline{r}', t) \, dV \right) dt \qquad \text{(where f(t) is a given function of time)}$$

by using the assumptions

(i) that the flux ϕ_g varies linearly with time

$$\phi_g(\underline{r}, t) \approx \left(\frac{t_p - t}{\delta t} \right) \phi_g(\underline{r}, t_{p-1}) + \left(\frac{t - t_{p-1}}{\delta t} \right) \phi_g(\underline{r}, t_p) \qquad (\text{where } \delta t = t_p - t_{p-1})$$

(ii) and that the cross-section σ_g is constant within the time step

$$\sigma_g(\underline{r}', t) = \sigma_g(\underline{r}', \bar{t}) \quad ,$$

where $\bar{t}$ is an average time for the step (not necessarily the mean).

We incorporate the assumptions into an $e^{\lambda_i t}$ weighted form of the precursor concentration equation and a directly integrated form of the diffusion equation [Stacey 1969]. After some elimination we obtain an equation for the flux at time t_p in terms of the flux and precursor concentrations at time t_{p-1}. To eliminate the $-\nabla \cdot D_{n.g}\nabla$ operator we resort to a finite difference method which embodies the required internal boundary conditions [Wachspress 1966]. The equations may then be written as

$$(L + (A + a) - (R + \frac{F}{k}))\phi(p) = -(L + (A - a) - (R + \frac{F'}{k}))\phi(p - 1) + 2HC(p - 1) + S(p, p - 1) ,$$

where $\phi(p)$ is a vector of unknown flux elements (involving space points as well as energy groups) at time t_p,
$\phi(p - 1)$ is a vector of known flux elements at time t_{p-1},
$C(p - 1)$ is a vector of known precursor concentrations (involving space points as well as delayed groups) at time t_{p-1},
$S(p,p - 1)$ is a vector of time integrated external source elements,
k is the steady state multiplication constant.

The remaining quantities are matrices derived from the indicated terms:

$$L \sim -\int_{box} \nabla \cdot D_{n.g} \nabla \, dV ,$$

$$A \sim \int_{box} \sigma_{rem.g} \, dV ,$$

$$a \sim \int_{box} 2/(v_g \delta t) dV ,$$

$$R \sim \int_{box} \sigma_{s.g' \to g} \, dV .$$

$$F \sim \chi_{(2)g}(\delta t) \int_{box} \upsilon \sigma_{f.g'} dV ,$$

$$F' \sim \chi_{(1)g}(\delta t) \int_{box} \upsilon \sigma_{f.g'} dV ,$$

$$H \sim \chi_{dg}(\delta t) ,$$

where $\chi_{(1)g}(\delta t)$, $\chi_{(2)g}(\delta t)$ and $\chi_{dg}(\delta t)$ are obtained from the fission spectra and delayed neutron data [Barry and Pollard, 1987 revised 1995].

In brief, the calculation of $\phi(p)$ for a (steady state) source problem consists of the following steps:

(i) group rebalance to balance neutron events for all groups,

(ii) MINI used to solve region rebalance to balance neutron events for the whole reactor for one group,

(iii) normally MINI, z, between planes, inner iterations for solution of one group for the whole reactor,

(iv) normally MINI, (x,y), within plane, inner iterations for solution of one group of each successive plane of the reactor, and

(v) normally MINI enhanced Gauss-Seidel upscattering and fission iterations for all groups and all space points.

For a steady state eigenvalue calculation step (v) is replaced by the following steps:

(v) normally MINI enhanced Gauss-Seidel upscattering iteration (fission is not included here), and

(vi) Chebyshev extrapolated outer iteration fission source calculation.

4. COMPUTING ENVIRONMENT

POW3D has been successfully ported from IBM/MVS to UNIX on the Fujitsu VP2200 supercomputer (as well as DOS on a PC). Most of the routines are written in FORTRAN 77 and compile with the Fujitsu **frt** compiler, using optimisation and vectorisation, which produces good object coding. The routines which are most important for efficiency of computing are those used in the inner loop MINI, ICCG or SLOR schemes for the (x,y) or (r,z) plane.

The entire program fits into less than 3 Megabytes of storage. However POW3D uses whatever further storage is available by virtue of its own virtual input/output routine, VIRTUL, [Appendix C] which runs under the operating system. For very lengthy 3D calculations the availability of say 30 Megabytes of storage can considerably reduce the elapsed time taken by the code. In any case many files are likely to be involved for both regular and temporary input/output.

Typical jobs (if these exist) usually take about the following CPU times on the Fujitsu VP2200 but a factor 4 (or thereabouts) increase or decrease is possible.

0&1D	steady state calculations	- a fraction of a second,
2D	steady state calculations	- a few seconds,
3D	steady state calculations	- half a minute,
2D	kinetics calculations	- a few minutes,
3D	kinetics calculations	- half an hour.

At ANSTO, POW3D may be run on the Fujitsu VP2200 supercomputer under the AUS scheme which is invoked with the **aus** (Bourne shell) script. For standardisation, and ease of use, when user supplied routines are to be added, the **makemakepow3d** (C shell) script is used to generate a **Makefile** to be used with **make** to create a temporary load module (Appendix B). All execution is then under AUS. Data required for a POW3D run can be broken down into 4 consecutive parts:

(1) occasionally required user supplied additional routines coded in FORTRAN.

(2) **aus** invocation using the Bourne shell script,

(3) AUS system control information [Robinson 1987, 1991] and

(4) POW3D data proper.

Most applications of POW3D do not require (1) and only standard usage of (2). For further specialised information on (1), (2), (3) see Appendices A, B and F.

With the **aus** invocation the following environment variables, which pass information to POW3D, may be set.

TIME	the maximum time (minutes) the job should execute
REGION	the maximum memory (Megabytes) the job should use
PLOT	the device on which plot output will eventually be plotted
PLOTFILE	the file on which plot output is to be written.

Also with the invocation of **aus**, particular cross-section, flux, geometry or other disk data files may be assigned if the defaults, mostly temporary disk data files, are not suitable. See Appendix A for the association of the disk data files with FORTRAN units in POW3D.

Examples of UNIX shell scripts and coding required for POW3D runs at ANSTO are tabulated below. For the temporary POW3D update feature only sections which differ from standard usage are listed.

Comment	Standard usage	Temporary POW3D update feature
FORTRAN		User supplied subroutines (Section 8) requiring variable dimensions should be stored in the working directory in files with names of the form *routine*.u . The user should execute the following two consecutive commands in the working directory (Appendix B2.2). **makemakepow3d** - to create **Makefile**, **make** - to create **pow3d**.
AUS invocation		
	aus myjob << 'eof'	
or, if required, flux dump file /dir/filename		
	aus myjob fl3=/dir/filename << 'eof'	
or cross-section file /dir/xsdata and flux file		
	aus myjob xs1=/dir/xsdata \ fl3=/dir/filename << 'eof'	
or set limits on memory (Megabytes) time (minutes)		
	REGION=30 TIME=20 aus myjob << 'eof'	
or produce plots (Section 6.26)		
	PLOT=delay PLOTFILE=myplot aus myjob << 'eof'	
AUS data	*dd1 step * link pow3d end stop *dd2	*dd1 step * testm pow3d=./pow3d link pow3d end stop *dd2
POW3D data		
	prelude ...	
see Sections 6,7		
	stop	
AUS termination	eof	

5. INPUT DATA STYLE

5.1 Conventions

In this report we adopt the following conventions:

- code names are given in UPPER CASE,
- input in general is in lower case,
- information to be reproduced exactly is given in **bold**,
- items that the user will replace with an actual value are given in *italics*,
- a data block, consisting of keywords and data items, is given as a *descriptive phrase in italics*,
- omission of some data is indicated by ... ,
- examples and defaults are given in `constant width` type,
- optional items are given inside [ ],
- items from which the user will choose are listed in a column inside { }.

5.2 Free Style (SKAN) Input to POW3D

Input data are supplied in free format style with lower case keywords indicating the data type, *e.g.*

```
sp 0.8  0.2  0.  0.
```

where the keyword **sp** designates that fission spectrum data follow. The data are read using the set of subroutines SKAN [Pollard 1978, also Appendix B.]. The data features are listed below and apply for both integer and floating point quantities except where otherwise stated.

DATA FEATURES	COMMENTS
Record columns	Data are entered in columns 1 to 72 only. (Each data record is listed when read but `$opt 1,72,0$` anywhere in the data beyond **prelude...end** may be used to turn off the listing option).
Keywords	May be up to 8 alphanumeric characters except that to avoid confusion zero is never used, *e.g.* the last character in **accfo** is the letter o.
Data	Floating point data may be given in abbreviated form, *e.g.* `1  1.  1.-0  1.e+0  1.e 0  1.d-0` are all equivalent. Alphanumeric data may consist of up to 8 alphanumeric characters, starting with an alphabetic character, and must not form keywords.
Extent of data	Different keywords require different extents of data to trail them, *e.g.* for a 4-group problem the fission spectrum, **sp**, would require 4 numbers to follow.
Data repeats	`n*d` means that the data entry `d` is repeated `n` times *e.g.* `4*0.` is the same as `0. 0. 0. 0.`

Data increments	`f(i)t` means start from `f` and go to `t` in increments of `i` *e.g.*
	`1(2)7` is the same as `1 3 5 7`
	The same mode must be given for `f`, `i` and `t`, *e.g.*
	`1.(0.5)3.` but not `1(0.5)3`

Readability	May be enhanced by use of special characters, *e.g.* `m(1)=0.07` is equivalent to `m 1 0.07` .
Comment records	A record with an * in column 1 is a comment and is not processed by SKAN.
Appended comments	Data following ## on an input record is a comment and is ignored, *e.g.* `sp=0.7,0.3,2*0 ## fission spectrum.`

6. STEADY STATE CALCULATIONS

6.1 Introduction

In section 2 the equations solved by POW3D were introduced. Here we detail the data requirements for a steady state calculation. The steady state calculation may be carried out as:

(1) the first stage of a kinetics study or

(2) part of routine reactor assessment.

The general (real) steady state equation solved by POW3D at each mesh point for the flux in a box about each point may be written as,

$$-\int_{box} \nabla \cdot D_{n.g}\, \nabla\phi_g dV + \phi_g \int_{box} \sigma_{rem.g} dV$$

$$= \sum_{g'} \phi_{g'} \int_{box} \sigma_{s.g'\to g} dV + \chi_g \sum_{g'} \phi_{g'} \int_{box} \frac{\upsilon}{k} \sigma_{f.g'}\, dV + S_g , \qquad \text{for } g = 1, 2, \ldots, ng,$$

where k is the effective multiplication constant (=1 for source problems),

$S_g(\underline{r})$ is the group g external source in each box (=0 for eigenvalue problems),

χ_g is the group fission spectrum (normalised to unit sum) and

$= \chi_{p.g}(1-\beta) + \sum_i \chi_{i.g}\,\beta_i$ the equilibrium spectrum for (1), with $\chi_{p.g}$ the prompt neutron spectrum and $\chi_{i.g}$ the spectrum for delayed neutron group i, yielding fraction β_i of total emissions $\beta (= \sum_i (\beta_i))$ of which are delayed, or

$= \chi_{p.g}$ the (normal) prompt spectrum for (2),

$\sigma(\underline{r})$ denotes various (macroscopic) cross-sections,

$\sigma_{s.g'\to g}(\underline{r})$ the scattering matrix,

$D_{n.g}(\underline{r})$ denotes possibly directional diffusion coefficients (for direction n or -n parallel to the chosen axes),

$\phi_g(\underline{r})$ is the group g flux in each box required to be calculated,

$\underline{r}$ denotes the coordinates of a typical mesh point (or a point in the box).

Sums are taken over all groups (1, 2,....,ng for neutron energy groups and 1, 2,....,igd for delayed neutron groups). The equations are solved for ϕ_g subject to either zero flux or zero current outer boundary conditions.

Similarly for adjoint flux ϕ_g^* the steady state equation (with corresponding boundary conditions) is

$$-\int_{box} \nabla \cdot D_{n.g}\, \nabla\phi_g^*\, dV + \phi_g^* \int_{box} \sigma_{rem.g}\, dV$$

$$= \sum_{g'} \phi_{g'}^* \int_{box} \sigma_{s.g\to g'}\, dV + \sum_{g'} \chi_{g'}\, \phi_{g'}^* \int_{box} \frac{\upsilon}{k} \sigma_{f.g}\, dV + S_g^* , \qquad \text{for } g = 1, 2, \ldots, ng.$$

The usual POW3D problems may either:

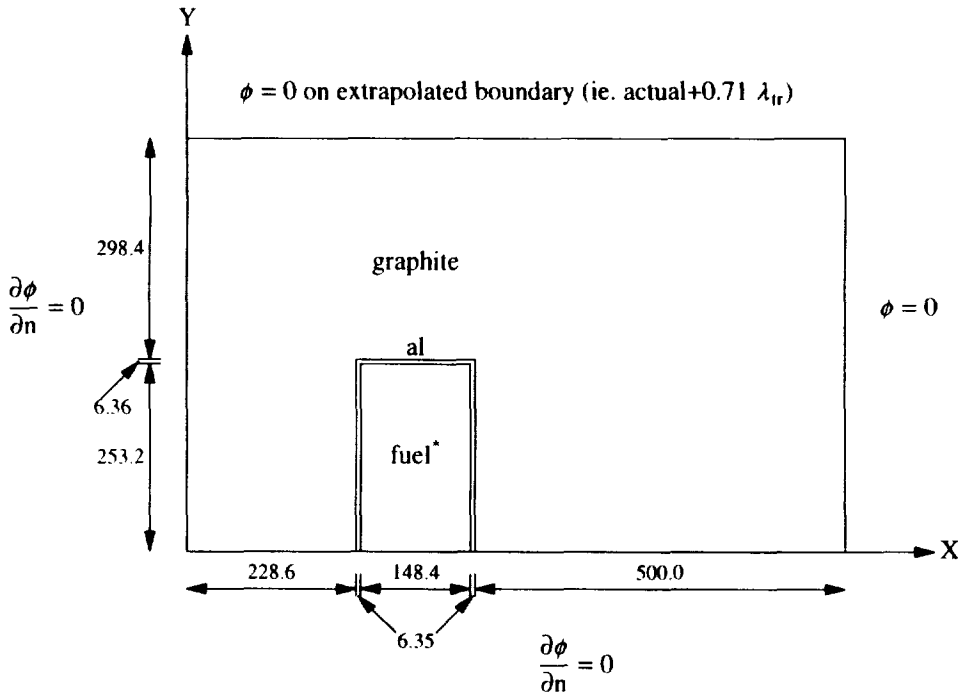
(a) calculate k as the eigenvalue of the homogeneous (external source free) equation, or

(b) search for an eigenvalue, λ, usually multiplying a physical quantity such as mesh spacing, to achieve a required value of k , k_{reqd}, (default 1); the corresponding eigenvector (for the **real** case normalised to a total fission source of unity) is then ϕ_g.

We start the main presentation of data for POW3D with a sample run. Next we introduce data for different applications as data blocks under headings such as 'geometry data'. An entire run consists of a combination of data blocks for a particular application.

6.2 A Sample Run (moata3d)

The Argonaut type reactor MOATA at ANSTO consists of highly enriched uranium plate fuel. It is light-water cooled and moderated and has mainly been operated at a power of 100 kW thermal. A major difference with some other Argonaut reactors is the replacement of the annular core by a system of two slab aluminium core tanks separated by 457.2 mm of graphite. The core tanks are graphite reflected (horizontally) and light water reflected (vertically). A plan view of homogenised regions of a quarter core model of MOATA with reflected boundaries is shown below with all dimensions given in millimetres (although POW3D uses centimetres). The input data for the sample run follow and are based on data generated under AUS [Robinson, 1987 and 1991]. Details for running the job are expanded in Appendix F. For a more current application see Robinson [1991a] which outlines calculations of the 10 MW reactor HIFAR.



(fuel* for $0 \le z \le 292.1$ and H_2O for $292.1 \le z \le 650.9$
reflective boundary at $z = 0$ and reactor boundary at $z = 650.9$)

MOATA REACTOR PLAN (CENTRE PLANE)

```
aus moata3d << 'eof'                                                    - AUS invocation
*dd1                                                                    - AUS data
step*
      link pow3d
      end
stop
*dd2                                                                    - POW3D data
prelude maxx=23,maxy=17,maxz=17,maxg=4,maxm=4 end - array sizes different than default
pow3d moata3d xyz calculation                     - descriptive data
ng=4                                              - number of groups
xsd fuel m(1)                                     - macroscopic cross-sections (cm-1)
1.54481-1 3.6863-1 9.02355-1 2.00955              - σtr (transport)
9.41612-2 6.02354-2 6.81026-1 1.12777-1           - σrem (absorption + scatter)
3.73274-4 3.30167-3 3.96448-2 9.77554-2           - νσf
0.0 9.37282-2 2*0.0                               - σs,1→i (group 1 to i outscatter)
2*0.0 4.88109-2 8.75701-3                         - σs,2→i
0.0 5.09412-3 0.0 6.48822-1                       - σs,3→i
0.0 1.66342-7 4.77318-2 0.0                       - σs,4→i
xsd al m(2)
1.16556-1 1.05169-1 9.05011-2 9.34386-2
1.64155-2 1.20851-3 1.84524-2 1.34341-2
4*0.0
0.0 1.6132-2 2*0.0
2*0.0 5.06778-4 0.0
0.0 5.36885-4 0.0 1.28382-2
2*0.0 1.27736-3 0.0
xsd graphite m(3)
1.49824-1 3.44423-1 3.74146-1 3.63541-1
2.24931-2 4.23208-3 9.82699-2 8.3279-3
4*0.0
0.0 2.24898-2 2*0.0
2*0.0 4.22309-3 4.23284-8
0.0 1.58544-3 0.0 9.65788-2
2*0.0 8.07923-3 0.0
xsd h2o m(4)
1.70681-1,4.42615-1,1.07516,2.25085
0.1242947,7.007271-2,0.8337886,7.457105-2
4*0
0,1.24079-1,2*0
2*0,5.90997-2,1.00901-2
0,3.44408-3,0,8.20634-1
0,1.78214-7,5.17295-2,0
sp 0.753564 0.246436 0 0                          - fission spectrum
xm=0.,5*4.572,0.635,5*2.968,0.635,10*5,0.71       - mesh spacing and first and
ym=0,5*5.064,0.636,10*2.984,0.71                    last boundary conditions
zm=0,6*4.86833,10*3.588,0.71
reg mx=1(1)22 my=1(1)16 mz=1(1)16 m(3)            - layout region definitions
reg mx=6(1)12 my=1(1)6 m(2)                         overlay each other
reg mx=7(1)11 my=1(1)5 mz=1(1)6 m(1)
reg mz=7(1)16 m(4)
start                                             - start calculation
groups 4  1,1(1)4                                 - edit requirements
mreg 1,2,3,4 -1=1,2,3,4
edit 1 1 6 0
stop                                              - stop POW3D run
eof                                                             - AUS termination
```

6.3 Overview of the Data Blocks

This section contains an overview of the data blocks required for a steady state flux calculation. When calculating the steady state conditions prior to a kinetics calculation the code uses an equilibrium spectrum. It is for this reason that the neutron data for kinetics (6.10) are included with the steady state data. In subsequent sections each data block is discussed in detail.

The data blocks are listed below essentially in the order in which data are required for a flux calculation. The data blocks with data essential for most POW3D steady state calculations within AUS are identified by **. Commands which trigger the calculation are identified by ***. Data blocks frequently used for the editing of output are identified by *.

- **(6.4) **prelude** data - to set array sizes for each independent calculation in a POW3D run,
- (6.5) Descriptive data - POW3D job name and heading,
- **(6.6) Neutron data from an AUS cross-section file - usually macroscopic but could be microscopic cross-section data,
- (6.7) Neutron data in the input stream - either microscopic or macroscopic cross-section data,
- (6.8) $P_n(n\neq0)$ data - Legendre polynomial weighted scattering data for use by other AUS modules,
- (6.9) Directional diffusion coefficients - material and energy dependent data,
- (6.10) Neutron data for kinetics - group velocities, fission energy release, delayed neutron data,
- (6.11) Cross-section modification - option to change selected cross-sections,
- (6.12) Material definition - data to produce cross-sections for mixtures of materials,
- (6.13) DB^2 leakage - data for setting up approximate leakage for 0, 1 and 2 D calculations,
- (6.14) Poison absorption - inclusion in removal cross-sections,
- **(6.15) Geometry data in the input stream - mesh spacing, boundary conditions and region layout,
- (6.16) Geometry data on disk - options to read and write an AUS geometry file,
- (6.17) Calculation type - real or adjoint flux; eigenvalue, source (or kinetics problem),
- (6.18) Criticality search data - code adjustment of data to achieve criticality,
- (6.19) Fixed external source specification - data for source calculation,
- (6.20) Trial flux specification - normally not required,
- (6.21) Coarse Mesh for Region Rebalance - normally not required,
- (6.22) Calculation termination conditions - accuracy and machine time limits,
- *(6.23) Output requirement specification - to set content of output,
- ***(6.24) Calculation **start** - or **restart** from a previous flux dump,
- *(6.25) Region **edit** - print or disk output of reaction rates or cross-sections for regions of materials,
- *(6.26) Point **edit** - print or plot, flux or reaction rates at mesh points,
- *(6.27) Channel power **edits** - print flux or reaction rates averaged over reactor channels,
- (6.28) Perturbation analysis - for small data changes using real and adjoint flux,
- **(6.29) Termination command - **end** and **stop**.

Usually geometry data are specified in the input stream (6.15), but could equally well be read from an AUS geometry file (6.16).

For a POW3D run within AUS, it is usual to read neutron data from an AUS cross-section file (6.6). However if an AUS cross-section file is not available, neutron data may be entered directly into the input stream (6.7). Another option, used in the early days of POW3D, selects data from a resonance library (with cross-sections tabulated as a function of potential scattering) such as the ABBN library [Bondarenko 1964] or the Hansen-Roach [1961] library. This option is no longer used but is retained nevertheless (Appendix E).

6.4 prelude Data

Prelude data set the dimensions of arrays for a POW3D calculation and must precede all other data. A POW3D run may consist of several independent calculations, each of which begins with **prelude** data and finishes with a termination command (Section 6.29). If the default array dimensions, listed in the table below, are not satisfactory the user must supply data between the keywords:

prelude [*dimensional data*] **end**

Usually a POW3D run requires only one **prelude** statement, for example:

prelude **maxx**=*23*,**maxy**=*17*,**maxz**=*17*,**maxg**=*4*,**maxm**=*4* **end**
data for POW3D calculation
stop

Storage may be assigned for two independent calculations within one POW3D run as follows:

prelude **maxx**=*23*,**maxy**=*17*,**maxz**=*17*,**maxg**=*4*,**maxm**=*4* **end**
data for first POW3D calculation
end
prelude **maxx**=*15*,**maxy**=*17*,**maxz**=*8*,**maxg**=*2*,**maxs**=*3*,**info**=*-1,0,0* **end**
data for second POW3D calculation
stop

prelude	Default	Description
maxx=25	81	Maximum number of x (or r) mesh points - since flux is calculated at mesh points, **maxx** must then be at least 1 plus the number of mesh intervals
maxy=27	1	Maximum number of y (or z) mesh points (in r-z geometry)
maxz=20	0	Maximum of z mesh points
maxg=4	18	Maximum number of groups
maxs=1	3	Maximum number of additional reactions (Section 6.7.2)
maxm=5	10	Maximum number of materials
maxf=1	0	Maximum number of fixed source or adjoint flux arrays (1 is required for a fixed source or kinetics problem, or for an adjoint flux problem if the real and adjoint flux arrays, **flx** and **fsce** respectively, are both required for **edit**)
maxw=0	400	Maximum length of work array **wsea** used with **search** (Section 6.18)
maxn=10	100	Maximum length of **mreg** array with **edit** (Section 6.25)
maxgd=6	0	Maximum number of delayed neutron groups
maxp=150	100	Maximum storage for data associated with description of kinetic pulses (Section 7.5.4)
maxdsk=1	2	If **maxdsk**=1 is specified then the virtual input/output (Appendix C) does not use memory buffering and all input/output is direct to disk
maxsct=1	2	If **maxsct**=1 is specified then the buffer used to store extensive volume integrated scattering data - **scatv** for all z-planes and groups - is not used and data goes to disk
info=-1,0,0	-2,2,2	POW3D flux dump/**restart** information (Section 6.24)
mist=50	100	Maximum length of **list** used with an **output** printer dump (**pdump** - Section 6.23) and variables used by the data manipulation feature introduced in Section 8.1
maxdat=100	1000	Maximum storage for compiled DATRAN, data manipulation code
maxvar=10	500	Maximum length of **var** array used with DATRAN
printable		Causes printing of array sizes based on specified **prelude** data
end		Terminating word of **prelude** data

Note: All other common data are set to default values, mostly zero, except values set with the 'common' file on unit 4.

6.5 Descriptive Data

Descriptive data usually follow **prelude** data and are given with one record as follows:

pow3d *jobname* *Heading*

where *jobname* is a descriptive name of up to 8 characters used to identify the job,
Heading is any suitable heading of up to 52 characters used to label output.

Example:

```
pow3d moata Test calculation
```

6.6 Neutron Data from an AUS Cross-Section File

To read neutron data from an AUS cross-section file, the required material data must first be selected and then a particular AUS cross-section file chosen. The data should be given in the following order:

xsd, material and cross-section data selection,

homovr, homogeneous volume ratios - infrequently used,

read lib, the selection and reading of a particular AUS cross-section file.

6.6.1 xsd, material cross-section data selection

The required data may be selected as follows:

xsd [[[*matname*] *source*] *mod*] **m**(*m*) [[[*conc*] *homovr*] *temp*]

where *matname* if given and if a material called *matname* is available in the cross-section file then that material is used; otherwise the material in the m^{th} position on the file is selected and is given the 1- to 8-character *matname*,

source if given, selects a particular entry for the named material (which may not be unique to the file); otherwise the file recommended data for the material (with highest update number) is used,

mod if given, selects a corresponding file entry appropriate for this modification of data; otherwise the file recommended data for the chosen material and data source is used,

m material number in POW3D - also if *matname* is not specified or material *matname* is not available in the cross-section file, then the m^{th} material in the file is selected,

conc material *conc*entration to be applied to data from a cross-section file (default 1),

homovr homogeneous volume ratio - mainly used with fission spectrum weighting Section 6.6.4 (default 0),

temp *temp*erature in degrees Kelvin, now not used (default 300) - see Note (1).

Examples of neutron data selection from an AUS cross-section file follow.

```
xsd  fuel,150gu5,orig  m(1)
xsd  fuel,90gu5        m(2)
xsd  fuel,90gu5,mod1   m(12)
xsd  d2o               m(3)
xsd  inv455b           m(5)
xsd                    m(7)
```

Notes:

(1) Neutron data may also be obtained from a resonance library like the 26-group, 3-temperature resonance tabular ABBN library [Bondarenko 1964], or the 16-group, 1-temperature resonance tabular Hansen-Roach [1961] library. Since resonance libraries are no longer used, this option, is detailed separately in

Appendix E. To obtain resonance shielded data with this option a maximum of 3 numbers [*conc homovr temp*] follow the **m**(*m*) data. If more than 3 numbers trail the material number, then POW3D treats the data which follow as input stream neutron data (Section 6.7), since even a 1 group problem must have at least 4 cross-sections.

(2) For kinetics calculations data are required for the pseudo 1/v-absorber **inv455b** (with 2200 m sec^{-1} value of $10^8/2.2 \times 10^5 \approx 455$ barns) and may be obtained as in the example above. The 1/v absorption cross-sections for **inv455b** are then calculated by POW3D from the library entries for group-averaged velocities in cm/shake (cm/10^{-8} sec) and stored as removal cross-sections.

6.6.2 homovr, homogeneous volume ratios

Homogeneous volume ratios are infrequently used but they provide weighting for the fission spectrum of each material (if several spectra exist) to produce a single fission spectrum (Section 6.6.3). As each **xsd** keyword is encountered, the homogeneous volume ratios are either read into the **homovr** array (Appendix E) or set to default values. The keyword **homovr**, followed by one word of data for each of up to maxm materials, may be used to overwrite these values. For example:

```
homovr 0.1005,0.8995
```

6.6.3 read lib, library selection

Cross-sections for the selected materials are sought and read when the following data are encountered:

read [*name*] **lib** *unit* [*spwt*]

where *name* is an optional name for user information ignored by POW3D unless the name is **input** (Section 6.7),

unit (usually 10,11 or 18, see note 1) selects the required FORTRAN unit, and

spwt (default 1) selects the fission spectrum weighting (see note 3).

Example:

```
xsd fuel m(1)
xsd d2o m(2)
read lib on 10
```

Notes:

(1) POW3D, as run under the AUS scheme [Robinson, 1987], has three main FORTRAN units from which cross-section files prepared by other AUS modules may be read. They are the FORTRAN units 10,11 and 18, coupled by default to the AUS disk files **xs1**, **xs2** and **xs3** (Appendix A). The main AUS cross-section library on FORTRAN unit 8 (coupled by default to AUS disk file **lib**), is not usually accessed directly by POW3D unless a standard resonance cross-section file is used (Appendix E). Selection of the required file is made using a feature of the **aus** (Bourne) shell script, as in the examples below:

```
aus myjob xs1=/dir/mylib1 << 'eof'
  ...
aus myjob xs2=/dir/mylib2  xs3=/dir/mylib3 << 'eof'
  ...
```

(2) Data for at most **maxs** additional reactions of the AUS cross-section file (Appendix A2), ignoring anisotropic diffusion coefficients, are returned in the following order:

(i) σ_p, potential scattering,

(ii) σ_{tot}, total,

(iii) σ_f, fission.

If anisotropic diffusion coefficients are available on the AUS cross-section file, they are read and stored in the appropriate elements of **dcx**(*m*), **dcy**(*m*) and **dcz**(*m*) (Section 6.9).

(3) POW3D permits only one prompt fission spectrum **sp** and uses an average spectrum which includes fissile materials of importance. When several different spectra exist, weighting may be selected with the data item *spwt* as follows:

(i) *spwt*=0, **sp** data from the AUS cross-section file *unit* are not entered into memory,

(ii) *spwt*=+G (default 1)

$$sp_g = \sum_{m=1}^{maxm} \nu\sigma_{f,G}(m)\text{homovr}(m)\chi_{p,g}(m) \ ,$$

that is, each spectrum is weighted by a particular group (G) homogeneous fission emission cross-section for each material (say 1 for fast reactor studies, *ng* for thermal reactor studies),

(iii) *spwt*=-G, as above except that the sum is added to the already existing **sp** data (possibly set up on a previous reading of the cross-section file,

(iv) $|spwt| > ng$ (number of energy groups) then

$$sp_g = \sum_{m=1}^{maxm} \left(\int \sum_{G} \nu\sigma_{f,G}(m)\phi \, dr \right) \chi_{p,g}(m)$$

is used if, following a partial calculation of ϕ, the user reads the cross-section file again.

Normalisation of sp_g to unit sum is not carried out until:

(a) just prior to a flux calculation initiated with **start**, or

(b) just prior to an **edit** calculation.

6.7 Neutron Data in the Input Stream

This option may be used if neutron data are not available on an AUS cross-section data file or it may be used in conjunction with such a file. The neutron data necessary for a steady state calculation consist of multigroup neutron cross-sections for each material and a single prompt fission spectrum for all materials. The number of energy groups **ng**, must precede all other data in this block. Although the fission energy release for each material may be needed for editing, it is not necessary for a steady state calculation and hence is described in the kinetics section. The required neutron data items are detailed in the subsections below.

Input stream neutron data are usually read from the standard FORTRAN input unit (1 in the AUS scheme). However input to this block may be temporarily switched to another FORTRAN unit, usually 7 or 12. Subsequent data, which may include data other than cross-section data, are read from this unit until terminated by an end of file (or the word **endofile** or POW3D **end** or **stop** commands). The code then reads any further data from the standard input unit. The user may switch FORTRAN input units with the following data:

read input[(*section*)] **lib** *unit*, *print*

where *section* (which must not be a keyword name) is a *section* of input stream data detailed later, and the following data must be provided

unit switches to reading the input stream data from FORTRAN *unit* (usually 7),

print { 0 does not print input stream data, 1 prints input stream data.

If *section* is omitted, reading begins with the first record. However, if *section* is given, then the file is skipped until a heading record is located with **input**(*section*) starting in the first character. For example, the file might consist of the data:

```
input(test1)   ## note - beginning in column 1 and packed together
...            ## normal POW3D data
endofile       ## the actual word 'endofile' designating end of section, 'test1'
input(test2)   ## the start of a second section
...
               ## an actual terminating end of file, or the word 'endofile'
```

6.7.1 ng, number of energy groups

The number of energy groups, *ng*, is defined as below. It must precede all other data in this section which must be consistent with this value.

ng *ng*

Example:

```
ng=4
```

6.7.2 xsd, cross-section data

Cross-section data, either microscopic, barns, or macroscopic, cm^{-1}, may be entered in the input stream in CRAM layout [Hassitt 1962].

xsd [[[*matname*] *source*] *mod*] **m**(*m*)

$\sigma_{tr,1}, \sigma_{tr,2},\ldots,\sigma_{tr,ng}$	- transport
$\sigma_{rem,1}, \sigma_{rem,2},\ldots,\sigma_{rem,ng}$	- removal (absorption + outscatter)
$\nu\sigma_{f,1}, \nu\sigma_{f,2},\ldots,\nu\sigma_{f,ng}$	- fission emission
$\sigma_{s,1\to1}, \sigma_{s,1\to2},\ldots,\sigma_{s,1\to ng}$	- scattering given as outscatters from group 1
$\sigma_{s,2\to1}, \sigma_{s,2\to2},\ldots,\sigma_{s,2\to ng}$	- scattering given as outscatters from group 2
...	
$\sigma_{s,ng\to1}, \sigma_{s,ng\to2},\ldots,\sigma_{s,ng\to ng}$	- scattering given as outscatters from group *ng*
	- optionally followed by
$\sigma_{r1,1}, \sigma_{r1,2},\ldots,\sigma_{r1,ng}$	- additional reaction 1,
$\sigma_{r2,1}, \sigma_{r2,2},\ldots,\sigma_{r2,ng}$	- additional reaction 2, up to *maxs* additional reactions.

An example of neutron cross-section data for one material follows:

```
ng=4
xsd fuel m(1)
1.54481-1  3.6863-1  9.02355-1  2.00955
9.41612-2  6.02354-2  6.81026-1  1.1277-1
3.73274-4  3.30167-3  3.96448-2  9.77554-2
0.0  9.37282-2  2*0.0
2*0.0  4.88109-2  8.75701-3
0.0  5.09412-3  0.0  6.48822-1
0.0  1.66342-7  4.77318-2  0.0
```

Notes:

(1) $\sigma_{s,g\to g}$, g=1,2,...,ng. POW3D sets self scatter entries of the scattering matrix to zero.

(2) The additional reactions may be used to store any cross-sections required for edit or other purposes. However the AUS cross-section file convention for additional reactions is given in Appendix A2.

6.7.3 sp, prompt neutron fission spectrum

A single group-integrated prompt neutron fission spectrum for all materials is entered with the data:

sp $\chi_{p,1}, \chi_{p,2},\ldots,\chi_{p,ng}$

where $\chi_{p,i}$ are values of the fission spectrum, later normalised by the code to unit sum.

Example:

```
sp  0.753564,0.246436,2*0
```

Note: If the delayed neutron fission spectra are different to the prompt spectrum, then the code uses an equilibrium spectrum (Section 6.10.4) in the steady state calculation prior to a kinetics calculation.

6.8 P_n ($n \neq 0$) Data

Although POW3D uses only P_0 scattering matrices, P_1 weighted and higher order matrices, may be included as data for separate 'materials'. These 'materials' are not used in a POW3D calculation, but they may be group collapsed by POW3D for use by other AUS modules. P_n ($n \neq 0$) data may be either entered in the input stream (Section 6.7.2) or selected from an AUS neutron cross-section data file (Section 6.6.1). The usual data requirements for a material apply, except that P_n ($n \neq 0$) data are indicated with

xsd ditto, p*n*

and must follow the P_0 data for that material in ascending order of *n* (*ie.* `p1, p2, . . . ,`). For example, if P_1 and P_2 data are available and required for material `hpo1`, then they may be selected with the data:

```
xsd   hpo1          m(1)  0.066
xsd   ditto,p1      m(2)  0.066
xsd   ditto,p2      m(3)  0.066
```

Note: To select and read P_n ($n \neq 0$) data, from an AUS cross-section file, the *matname* 'ditto' must be used. For example, the purely positional type request (Section 6.6.1),

```
xsd  m(1)  0.066  xsd  m(2)  0.066  xsd  m(3)  0.066
```

cannot be used since P_1 data does not exist in the cross-section file as a separate material.

6.9 dcx(m), dcy(m), dcz(m), Directional Diffusion Coefficients

If the user does not specify directional diffusion coefficients then the code assumes that the coefficients are isotropic and given by,

$$dcx(m) = dcy(m) = dcz(m) = 1/(3\,\sigma_{tr,g}), \quad \text{for } g = 1,2,\ldots,ng,$$

where dcx(*m*) denotes the x-direction diffusion coefficients for material *m* given a group at a time, with a similar meaning for dcy(*m*) and dcz(*m*).

The user may specify anisotropic diffusion coefficients as follows:

dcx(*m*) $D_{x,1}, D_{x,2},\ldots,D_{x,ng}$

dcy(*m*) $D_{y,1}, D_{y,2},\ldots,D_{y,ng}$

dcz(*m*) $D_{z,1}, D_{z,2},\ldots,D_{z,ng}$

where *m* is the material number,

$D_{x,g}$ is the diffusion coefficient in the x-direction for group g,

$D_{y,g}$ is the diffusion coefficient in the y-direction for group g,

$D_{z,g}$ is the diffusion coefficient in the z-direction for group g.

Example:

```
dcx(1) = 2.19  0.9  0.36  0.16
```

Almost equivalent data for r-z (cylindrical) geometry may be supplied:

dcr(*m*) $D_{r,1}, D_{r,2},\ldots,D_{r,ng}$ - diffusion coefficients in the r-direction (=dcx(m)),

dcz(*m*) $D_{z,1}, D_{z,2},\ldots,D_{z,ng}$ - diffusion coefficients in the z-direction (=dcy(m)).

When **dcz** is used the code also stores transport cross-sections consistent with **dcz**, namely

$$\sigma_{tr,g} = 1/(3D_{z,g}) \quad .$$

Note: When using x-y (plane) geometry each material may have diffusion coefficients in 3 directions,

(i) $D_{x,g}$

(ii) $D_{y,g}$

(iii) $1/(3\sigma_{tr,g})$ used in a DB^2 type approximation of leakage in the z-direction (Section 6.13).

6.10 Neutron Data for Kinetics

For a steady state calculation preceding a kinetics calculation the following kinetics data are necessary so that an equilibrium fission spectrum can be used:

vel, average group velocities (cm sec^{-1}),

fer, the fission energy release for each material (joules/fission),

igd, the number of delayed neutron groups,

betad, β_i, yield fractions for delayed neutron group i per emitted neutron,

dlamda, λ_i, the decay constants for the precursors (sec^{-1}), and

sd(i), $\chi_{i,g}$, the delayed neutron fission spectrum.

Fission energy release for each material is included on AUS cross-section data files and need not be provided for those materials obtained from AUS data files.

6.10.1 vel, group velocities

The average group velocities in cm sec^{-1} may be supplied trailing the keyword **vel** as follow:

vel $v_1\ v_2 \ldots v_{ng}$

Example:

```
vel    2.5+9 3.5+7 7.5+5 2.25+5
```

Alternatively if the material *inv455b* (Section 6.6.1) is selected and the keyword **vel** has no trailing data then POW3D calculates its own group velocities using the formula:

$v_g = 10^8/\sigma_{rem,g}$ (*inv455b*)

Example:

```
xsd inv455b  m(5)
read lib on 10
vel
```

Note: If after a steady state calculation the energy groups are collapsed (Section 6.25.1) then an appropriate way to obtain average velocities would be to average the *inv455b* cross-sections over the whole reactor. The short list feature above may be subsequently used to pick up the reactor-averaged group velocities.

6.10.2 fer, fission energy release

The fission energy release for each material is needed to obtain reactor power for a kinetics calculation or for specialised editing. Fission energy release may be obtained from AUS cross-section files, or alternatively, supplied for each material in turn as follows:

fer $f_1, f_2, \ldots f_{maxm}$

where f_m is the fission energy in joules/fission for material m with default values of 3.172-11 for fissile and 0 for non-fissile materials.

Example:

```
fer    3.172-11, 2*0
```

6.10.3 delayed neutron fractions (betad) and precursor decay constants (lamda)

The number of delayed neutron groups, *igd*, must be defined first. The delayed neutron fractions and the precursor decay constants may then follow:

igd *igd*
betad $\beta_1, \beta_2, \ldots, \beta_{igd}$
dlamda $\lambda_1, \lambda_2, \ldots, \lambda_{igd}$

where *igd* is the number of delayed neutron groups (≤ **maxgd** specified in the **prelude**),
β_i are yield fractions for delayed neutron group i per emitted neutron,
λ_i are delayed neutron precursor decay constants (sec^{-1}).

Example:

```
igd=1
betad=6.4-3
dlamda=0.08
```

Notes:

(1) Kinetics calculations for prompt excursions may be carried out without delayed neutrons, (*ie.* `igd=0`, which is the default value).

(2) The total delayed neutron fraction $\beta = \sum_{i=1}^{igd} \beta_i$ is available as a weighting option for reactivity insertion (Section 7.6).

6.10.4 sd(i), delayed fission spectrum

The delayed neutron fission spectrum $\chi_{i,g}$, may be supplied for each of a total of *igd* delayed groups *i* producing neutrons in energy group *g*, as follows:

sd(*i*) $\chi_{i,1}, \chi_{i,2}, \ldots, \chi_{i,ng}$

To simplify data presentation the following interpretation of short lists is adopted:

sd(*i*) - without further data uses the immediately preceding **sd** data,
sd - without further data takes all the spectra to be given by the prompt values **sp**.

The minimal and usual delayed fission spectrum requirement for few group calculations, is the lone word **sd**. For many group calculations, in which the delayed fission spectra are different from the prompt spectrum, the user must select the fission spectrum required for editing. This is because when delayed data are supplied POW3D uses an equilibrium spectrum

$$sp_g = \chi_{p,g}(1-\beta) + \sum_{i=1}^{igd} \chi_{i,g}\,\beta_i$$

during the steady state flux calculation. POW3D normally restores the fission spectrum, sp_g, to the prompt values, $\chi_{p,g}$. However in editing, for consistency both with the steady state calculation and any subsequent kinetics calculation, the equilibrium spectrum should be used (`nsd=1`). The required fission spectrum may be selected with the keyword **nsd** (default `nsd=0`) before **start** thus:

nsd *n*

where *n* { 0 prompt spectrum is used in editing,
1 equilibrium spectrum is used in editing.

Notes:

(1) The data `nsd=1` must not be supplied more than once within a run, since it implies that the data in **sp** is a prompt spectrum and the code changes it to an equilibrium spectrum.

(2) The code normalises the spectra to unit sum prior to any calculation.

6.11 Cross-Section Modification

The main purpose of the feature is to enable the effect of data perturbations to be studied. Neutron cross-section modification may be either multiplicative or additive and uses numbers x, one corresponding to each reaction and each group of material *m*, as follows:

xsd modify m(*m*) [*f* **m**(*m*)]

$x_{tr,1}$, $x_{tr,2}$,...,$x_{tr,ng}$

$x_{abs,1}$, $x_{abs,2}$,...,$x_{abs,ng}$

$x_{vf,1}$, $x_{vf,2}$,...,$x_{vf,ng}$

$x_{s,1\to1}$, $x_{s,1\to2}$,...,$x_{s,1\to ng}$

$x_{s,2\to1}$, $x_{s,2\to2}$,...,$x_{s,2\to ng}$

...

$x_{s,ng\to1}$, $x_{s,ng\to2}$,...,$x_{s,ng\to ng}$

$x_{r1,1}$, $x_{r1,2}$,...,$x_{r1,ng}$ - followed by numbers for up to **maxs** optional additional reactions

$x_{r2,1}$, $x_{r2,2}$,...,$x_{r2,ng}$, *etc.*

The *f* **m**(*m*), data are optional and only necessary if the default value f = 1 is not satisfactory. It should be noted that the second group of numbers, $x_{abs,g}$, relates to the absorption rather than removals.

The cross-section modification procedure adopted is the following, where a prime denotes the modified data.

(1) $\sigma_{abs,g} = \sigma_{rem,g} - \sum_{g \neq g'} \sigma_{s,g'\to g}$, and this becomes reaction 2.

(2) If x=0, then the cross-section is unaltered, *i.e.* $\sigma' = \sigma$,

otherwise a modified value x′ is assumed, $x' = \begin{cases} x, & \text{for } |x| > 10^{-30} \\ 0, & \text{for } |x| \leq 10^{-30} \end{cases}$

and used to modify the cross-section data, $\sigma' = \begin{cases} |f|\sigma + x', & \text{for } f \leq 0 \\ f\sigma x', & \text{for } f > 0. \end{cases}$

(3) If a transport cross-section is modified then the directional diffusion coefficients are also modified *i.e.*

$D_{x,g} = D_{y,g} = D_{z,g} = 1/(3\sigma'_{tr,g})$.

(4) $\sigma'_{rem,g} = \sigma'_{abs,g} + \sum_{g \neq g'} \sigma'_{s,g'\to g}$, restoring reaction 2 to removal cross-section.

In the following example absorption for material m(2) in group 1 (of 16-group data) is increased by 1% :

```
xsd modify m(2) 16*0, 1.01, 15*0, 272*0
```

6.12 Material Definition

Mixing of either microscopic or macroscopic cross-section data, may be required to form data for mixtures or components of materials. Mixing is usually done immediately when definition data are encountered but may also be carried out as part of a criticality search (Section 6.18.3). The material definition data are of the form:

defn *matname* **m**(m_d) **m**(m_1) $conc_1$ **m**(m_2) $conc_2$ ...

where *matname* is a 1- to 8-character name for the defined material,

m_d is a material number for the defined material,

m_i is the material number of a previously entered or defined material,

$conc_i$ is the concentration of material m_i to be included as part of material m_d.

Consider for example, that microscopic cross-section data are available for hydrogen and oxygen on an AUS cross-section file on FORTRAN unit 10. The following instructions for data selection, library reading and material definitions could then be used:

```
xsd h m(1)
xsd o m(2)
read lib on 10
defn h2o m(3)  = m(1) 2 m(2) 1
defn water m(4)  = m(3) 0.033
```

The rules used to build the cross-sections follow. To obtain diffusion coefficients for the defined material, these rules are applied by POW3D to the reciprocal values of diffusion coefficients of the constituent materials.

(i) If $m_d \neq m_1$, then the cross-sections of the defined material, m_d, are initially cleared and the cross-sections of material m_1 (multiplied by the concentration $conc_1$) are added to the cross-sections of material m_d.
If $m_d = m_1$, then the defined material cross-sections are multiplied by the concentration of material m_1.

(ii) Similarly for the 2nd material, except that cross-sections are no longer cleared, and so on.

Using these rules alternative definitions for the example above could be written as follow:

```
defn  h2o   m(3) =  m(1) 2 m(2) 1
defn  water m(3) =  m(3) 0.033
```

6.13 DB^2 Leakage

Inclusion of DB^2 in the removal cross-section for each material is a standard way of making approximate allowance for leakage of neutrons in directions not otherwise accounted for in the calculation. For zero-dimensional calculations all leakage is included this way. DB^2 leakage data may be specified as follows:

bsq(n) [$l_1, l_2, \ldots, l_k$ $m_1, m_2, \ldots, m_k$ $B_1^2, B_2^2, \ldots, B_k^2$]

where n
- -ve DB^2 added to data in additional reaction $|n|$,
- 0 DB^2 added to removal cross-sections,
- +ve DB^2 stored in additional reaction n,

l_i material number for which DB^2 leakage is to be calculated,

m_i material number for transport cross-sections to be used,

B_i^2
- ≠0 is the buckling to be used, or if
- =0 group dependent bucklings for m_i provided with **gbsq** (Section 6.13.1),

k is the number of materials for which leakage is to be calculated.

The DB^2 for the material l_i is calculated using the buckling B_i^2 divided by 3 times the transport cross-section of material m_i and this is added to the removal cross-sections.

$$\sigma'_{rem,g} = \sigma_{rem,g} + B_i^2 / (3\sigma_{tr,g})$$

Example:

```
bsq(3)  =  1,2,3,4  1,2,3,4  1.-3,1.-3,1.-3,1.-3
```

or entered more compactly,

```
bsq(3)=1(1)4  1(1)4  4*1.-3
```

Additional reactions used by **bsq** (or **abs**) should first be set to zero using the **abs** feature (Section 6.14), *e.g.*

```
abs(3)=1(1)6  1(1)6  6*0
bsq(3)=1(1)4  1(1)4  4*1.-3
```

Notes:

(1) The value of **maxs** must be set in the **prelude** so that $n \leq$ **maxs** .

(2) After a criticality search (Section 6.18) critical leakage may be permanently associated with the removal cross-section rather than simply as an additional reaction added only during the criticality search. This may be achieved with the **bsq** option for $n < 0$ and no data trailing n. The cross-sections corresponding to the additional reaction number n are then added to the removal cross-sections of material l_i (m_i and B_i^2 on the previous **bsq** entry are not used). For example, cross-sections for additional reaction 3 may be added to the removal cross-sections using the data:

```
bsq(-3)
```

6.13.1 Group dependent leakage from each region

A more precise way to approximate leakage, than that discussed above, is to use group dependent bucklings for each material $B_g^2(m)$. These may be entered, together with **bsq** data, as follow:

gbsq	$B_1^2(1), B_2^2(1),\ldots,$	$B_{maxg}^2(1)$	- group bucklings for material 1
	$B_1^2(2), B_2^2(2),\ldots.$	$B_{maxg}^2(2)$	- group bucklings for material 2
	...		
	$B_1^2(\text{maxm}), B_2^2(\text{maxm}),\ldots,$	$B_{maxg}^2(\text{maxm})$	- group bucklings for material maxm.

For example, the following leakage data given in the indicated order,

```
prelude maxg=4 maxm=3 ...     end
 ...
gbsq 4*0.,1.2-3,0.72-3,0.11-3,-0.23-3,4*0.
bsq(3)  2  2  0.
```

would generate values in additional reaction 3 for material 2 using material 2 transport cross-sections as follow:

$$\sigma_{reac3,g}(2) = B_g^2(2) / (3\sigma_{tr,g}(2)), \quad g = 1, 2, \ldots, ng \qquad (ng \leq maxg).$$

The real value of this feature lies with the POW3D capability to calculate the group dependent bucklings for each material constituting the reactor following a flux (and possibly adjoint) calculation. The required data to generate group dependent bucklings are supplied within a single floating point number,

gbsq *l.ijk*

where $l =$ $\begin{cases} 0 \text{ - flux } (\phi) \text{ weighting only (adjoint flux, } \phi^*, \text{ set to 1 in equation below)} \\ 1 \text{ - } \phi \text{ and } \phi^* \text{ weighting,} \end{cases}$

i,j,k 0 for omission, 1 for calculation of leakage in x,y and z directions respectively.

POW3D calculates (in a finite difference formulation) the group dependent bucklings in the required direction for the material in each region (interpreted via **mxs**) as follows:

$$B_g^2(m') = 3\sigma_{tr,g}(m') \int_M \phi_g * \nabla \cdot D_{n,g}(m') \nabla \phi_g \, dV \Big/ \int_M \phi_g \phi_g^* \, dV$$

where M consists of all regions of the reactor filled by material m,

m' = mxs(m), allows the cross-sections of another material to be used (Section 6.25.3) although normally $m'=m$ and

$D_{n,g}(m')$ denotes the possibly directional diffusion coefficients of material m' (Section 6.9).

For example

```
gbsq  1.001
```

would result in the calculation of group dependent bucklings in the z direction only.

In the first example of this section the following data could be used instead

```
... start gbsq 0.001 bsq(3)  2  2 0 ...
```

6.14 Poison Absorption

Inclusion of poison absorption in the removal cross-section of each material

$$\sigma'_{rem.g} = \sigma_{rem.g} + \sigma_{abs.g} \text{ (poison)}$$

is a possible way of making a reactor just critical (Section 6.18.2). The feature is exactly analogous to DB^2 leakage of the previous section. Data are specified in a similar way except that the keyword **abs** is used and poison concentrations (including zero concentrations if needed) are entered instead of bucklings. For example the following data may be used,

```
abs(3)=1(1)4  4*5  4*1.-3
```

where m(5) is the poison material and 1.-3 is the concentration.

6.15 Geometry Data in the Input Stream

Usually geometry data are entered in the input stream as detailed in this section. Options are also available to read and write geometry data using an AUS GEOM file on disk (Section 6.16).

Geometry data specify the type of geometry, width of mesh intervals, boundary conditions on the flux ϕ_g and the layout of materials in regions. Each plane, such as an (x,y) or (r,z) plane, is divided into cells by lines perpendicular to the axes at the specified mesh intervals. (An appropriate extension is applied to 3D.) The flux calculated by POW3D is the edge flux for reactions about points of intersection of the mesh lines (rather than at the centre of a mesh interval).

The required geometry is specified by use of appropriate keywords (followed by the associated data) as listed below. Note that for (x,y,z) geometry the user must specify **maxz** > 1 in the **prelude**.

Keywords	Geometry		
xm... **ym**... **zm**...	3D	rectangular	(x,y,z)
xm... **ym**... (or **xm**... **zm**...)	2D	rectangular	(x,y)
rm... **zm**...		cylindrical	(r,z)
xm...	1D	slab	x (y degenerated to 1 point)
rm...		cylindrical	r (z degenerated to 1 point)
rm(sphere)...		spherical	r
rm(sphere)...	0D	spherical	(r reduced to 2 points)

6.15.1 Mesh intervals and boundary conditions

Mesh intervals and boundary conditions are specified with data following any of the keywords **xm**, **ym**, **zm**, **rm**, or **rm(sphere)** as appropriate.

xm d_L, δ_1, δ_2,..., δ_{nxmi} , d_R

where d_L is the left boundary condition (usually 0 or 0.71) and
δ_i is the width of the i^{th} mesh interval (cm) of a set of intervals,
nxmi is the number of mesh intervals (the number of mesh points is *nxmi*+1)
d_R is the right boundary condition (usually 0.71).

The boundary conditions are given as extrapolation lengths in units of transport mean free paths. Reflective conditions (zero current) require d_L or $d_R = 0$. For cylindrical or spherical geometry the left boundary condition is zero (**rm** 0 ...). The usual reactor conditions with zero flux on extrapolated boundary require d_L or $d_R = 0.71$, but 1.0-4 may be used to force $\phi_g = 0$ on the actual boundary.

An example of three-dimensional geometry, data for moata3d follows:

```
xm=0,5*4.572,0.635,5*2.968,0.635,10*5,0.71
ym=0,5*5.064,0.636,10*2.984,0.71
zm=0,6*4.86833,10*3.588,0.71
```

An example of one-dimensional geometry data for GODIVA, which is a critical ^{235}U sphere of radius 8.71 cm (Argonne National Laboratory 1963), follows:

```
rm(sphere)=0,10*0.871,0.8
```

In this case the extrapolation distance of 0.8 (instead of the usual 0.71) transport mean free paths gives better agreement with an equivalent transport calculation.

Notes:

(1) The choice of mesh spacing cannot be prescribed but the following rough guidelines are provided:

(a) physical regions, unless homogenised with neighbouring regions, need to have their boundaries along mesh interval boundaries,

(b) large mesh spacing (2 or 3 transport mean free paths for 'important' neutrons) should be avoided as estimation of spatial variation becomes inadequate,

(c) small mesh spacing (1/10 transport mean free path for 'important' neutrons) should be avoided as convergence of the iterative solution is needlessly slowed down.

(2) 0D calculations may be specified simply as a reflective sphere with the data

```
rm(sphere)=0,1,0.
```

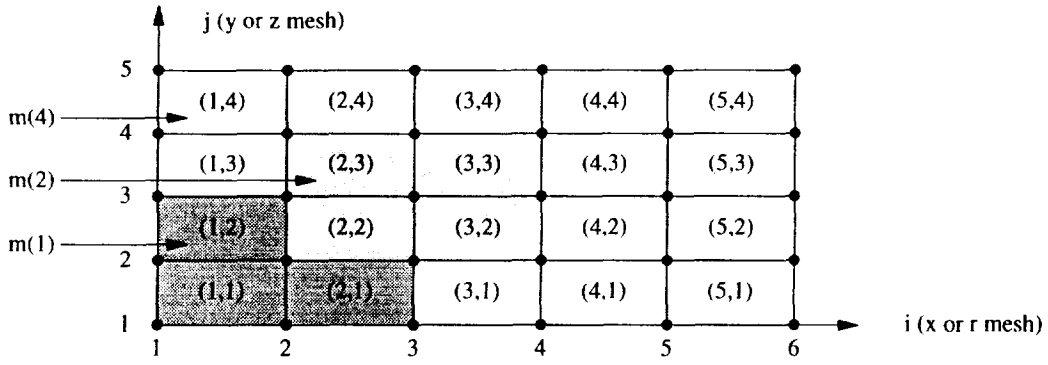
(3) 1D problems are specified with the fully reflective data in the y or z directions

```
ym = 0,2,0, or
zm = 0,2,0
```

where the mesh spacing of 2 is necessary if the volume is important. In the absence of special coding in POW3D the above data would specify 2 points for each x or r point. However POW3D exploits the degeneracy to 1 point which is particularly important since the inner line solution is thereby exact without any line relaxation.

6.15.2 reg, layout of materials in regions

The layout of constituent materials within the reactor is specified using the mesh cell numbering system illustrated below for one plane of the layout.



The layout of materials may be specified with a succession of statements of the type:

reg **mx** $i_1,i_2,...,i_I$ **my** $j_1,j_2,...,j_J$ **m** (m) - for 2D, or

reg **mr** $i_1,i_2,...,i_I$ **mz** $j_1,j_2,...,j_J$ **m** (m) - for 2D cylindrical, or

reg **mx** $i_1,i_2,...,i_I$ **my** $j_1,j_2,...,j_J$ **mz** $k_1,k_2,...,k_K$ **m** (m) - for 3D,

where i, j, k are interval numbers,

m is the number of the material which fills appropriate elements of the layout matrix.

The keyword **reg** need not be repeated for consecutive layout data items **mx**, **my** (and **mz**). Other data need not be supplied unless the data are changed. The layout illustrated above could be specified by first totally filling the matrix, and then superimposing materials on cells of the matrix as shown in the example below.

```
reg  mx = 1(1)5  my = 1(1)4  m(4)
```

$$\begin{pmatrix} 4 & 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 & 4 \end{pmatrix}$$

```
[reg]  mx = 1,2  my = 1,2  m(1)
```

$$\begin{pmatrix} 4 & 4 & 4 & 4 & 4 \\ 4 & 4 & 4 & 4 & 4 \\ 1 & 1 & 4 & 4 & 4 \\ 1 & 1 & 4 & 4 & 4 \end{pmatrix}$$

```
[reg]  mx = 2,3  my = 2,3  m(2)
```

$$\begin{pmatrix} 4 & 4 & 4 & 4 & 4 \\ 4 & 2 & 2 & 4 & 4 \\ 1 & 2 & 2 & 4 & 4 \\ 1 & 1 & 4 & 4 & 4 \end{pmatrix}$$

Notes:

(1) Clearly no part of the matrix should remain unfilled.

(2) 3D data specifications are an obvious extension - *e.g.* the moata3d sample run (Section 6.2).

(3) 1D **reg** data specifications do not require **my** (or **mz**) information, *e.g.*

```
reg  mx = 1(1)5  m(1)  mx = 6(1)10  m(2)
```

(4) 0D data specifications require only the material number, for example

```
reg  m(3)
```

6.15.3 Transform of mesh intervals

The transform option can be used to increase or change the number of mesh intervals in existing geometry data and associated flux arrays. For complex geometries this option can facilitate changes to finer or coarser mesh spacing if required. Transform data must not precede region layout data. The data requirement is:

mx nx_1, nx_2 ....,nx_n ....,nx_i
my ny_1, ny_2 ....,ny_n ....,ny_j
mz nz_1, nz_2 ....,nz_n ....,nz_k
transform

or for cylindrical geometry

mr nr_1, nr_2 ....,nr_n ....,nr_i
mz nz_1, nz_2 ....,nz_n ....,nz_j
transform

where nx_n $\begin{cases} \text{+ve} & \text{is the number of equal mesh intervals to replace the } n^{th} \text{ x-mesh interval,} \\ 0 & \text{the } n^{th} \text{ x-mesh interval is deleted and added to the succeeding region,} \end{cases}$

and i ≤ *nxmi* where *nxmi* is the number of x-mesh intervals in the existing geometry.

In other directions ny_n , nz_n , nr_n data have similar meanings to nx_n data. If a flux or adjoint flux array exists then fluxes are also interpolated and/or deleted to produce arrays consistent with the *transform*ed geometry.

In the example below the layout and mesh spacing on the left followed by the *transform* instructions produce the layout and mesh spacing on the right.

$$\begin{pmatrix} 4 & 4 & 4 & 6 \\ 2 & 3 & 4 & 5 \\ 1 & 2 & 4 & 7 \end{pmatrix}$$

```
xm = 0,1,0.75,0.5,2.0,0.71
ym = 0,1,0.5,2.0,0.71
reg ...
mx = 2,1,0,2 my = 2,2,1
transform
```

$$\begin{pmatrix} 4 & 4 & 4 & 6 & 6 \\ 2 & 2 & 3 & 5 & 5 \\ 2 & 2 & 3 & 5 & 5 \\ 1 & 1 & 2 & 7 & 7 \\ 1 & 1 & 2 & 7 & 7 \end{pmatrix}$$

```
xm = 0,2*0.5,0.75,2*1.25,0.71
ym = 0,2*0.5,2*0.25,2.0,0.71
```

6.16 Geometry Data on Disk

Generally geometry data are provided in the input stream (Section 6.15) but options are available for reading and writing geometry data on disk (Appendix A). Geometry data produced by an AUS module may be read from FORTRAN unit 26 with the data

read reg on 26

As an extension of the AUS geometry file, POW3D can have several named geometries for different jobs held in the one file. Named entries may be recalled with the data,

read *name* **reg on** 26,1

where the trailing 1 selects the option that reads through the entire file. The last appearing data identified by *name* is used. If, on the other hand, the trailing number is -1 then only the layout matrix is read. This latter feature is mainly for complicated usage of **edit** where the layout matrix is changed.

AUS geometry data may be written on FORTRAN unit 26 with the data

write reg on 26

If an AUS cross-section file has not been written in this run or if **mreg**=0 then the geometry data are written as specified. Otherwise material numbers for this geometry file are changed and refer to the last AUS cross-section file which was written. To write the first named data in a file or to write at the beginning of a file the following data must be used,

write *name* **reg on** 26

Subsequent named entries may be written with the data,

write *name* **reg on** 26,1

where the trailing 1 selects the option that writes the named data at the end of a geometry file.

6.17 Calculation Type

The required type of calculation may be chosen from the options below.

$$\textbf{calc} = \left\{\begin{array}{lll} \textit{real}\ , & \textit{eigenv}\ , & \textit{regcode} \\ \textit{adjoint}\ , & \textit{source}\ , & \ldots \\ & \textit{kinetics}\ , & \end{array}\right\}$$

The default values (*real, eigenv, regcode*) correspond to the normal code requirement. The (*adjoint, kinetics*) combination is not allowed. The regular code option, *regcode*, is normally used. However a feature is available in the code to accept different calculations for the coefficient matrix. One example used locally is specified with the option *hifpow3d* instead of *regcode*. In this case the HIFAR reactor model of Robinson [1991a] is used for the coefficient matrix which includes coarse control arm representation.

Example:

data for a steady state calculation

```
calc=real,eigenv
start
```

data for a kinetics calculation

```
calc=real,kinetics
start
```

Notes:

(1) As in the example above, a kinetics calculation would usually need to be preceded by a steady state calculation (default option and need not be specified). The kinetics option should be selected after the steady state calculation (*i.e.* after the **start** keyword).

(2) If storage is made available for the fixed source array **fsce** during an adjoint eigenvalue calculation by setting **maxf**=1 in the **prelude** then the adjoint solution is stored in the fixed source array. Both the real and adjoint fluxes may then be used concurrently for edit purposes.

6.18 Criticality Search

Three types of criticality search are currently available. These vary an eigenvalue λ, which normally multiplies some physical quantity, to achieve a required effective multiplication constant $k=k_{reqd}$ (usually 1). The searches are controlled by the routines **sub***n*, n=1,2,...,9, where only the first three are currently dedicated to specific tasks. Users may write their own routines in FORTRAN and load them with the temporary update feature (Section 4 and Appendix B). Users may also write routines in DATRAN, Section 8.1.

The available criticality search routines are:

sub1 - a general concentration search routine which may either adjust the concentrations of poison materials or of general materials,

sub2 - a general mesh width adjustment routine which may make simple changes of scale to all mesh intervals as well as more complicated data changes associated with differential and coupled adjustment,

sub3 - a general region adjustment routine which may move material in or out of regions of the reactor either in single channels or coupled channels according to a specified schedule.

In the next section the following requirements, which are common to all the subroutines are outlined.

search - data to select the required search option,

wsea - additional data required by the search subroutines,

fsea - infrequently required eigenvalue parameter data,

lsea - minimum limits on number of iterations.

6.18.1 Criticality search data

The main data requirement is associated with effective multiplication parameters as follows:

search$(x) = k_{reqd}$, k_{acc} , k'_{acc} , h

where x { 0 (the default option) then no search is carried out,
$\neq 0$ then the option parameter x is interpreted as below,

k_{reqd} is the required effective multiplication (default value 1.),

k_{acc} is the required accuracy in k_{reqd} such that for convergence $| k - k_{reqd} | \leq k_{acc}$ (default value 1.-3),

k'_{acc} is the required accuracy in k_{reqd} before source extrapolation is permitted (default value 1.-2) - not used with all options, and

h is an exponent used in the interpolation/extrapolation such that chord approximation of k^h as a function of critical eigenvalue λ is as reasonable as possible (h=-1.0 is best for variation of purely absorbing materials (poison search): default value 1.).

The option parameter x is of the form $x = si.d_1d_2d_3d_4$

that is s = sign,
i = integer part of x,
d_1 = first fractional digit, and
d_2 = 2nd fractional digit, etc.

where s { +ve λ updated every fixed number of outer iterations regardless of the source convergence (set with **lsea** data l_1),
-ve λ updated after partial source convergence (set with **accfo** - Section 6.22.1),

i available for use by some search subroutines (Section 6.18.2),

d_1 { $= 0$ λ_{n+1} or λ^*_{n+1} (**aeigen**) is calculated for use in **sub**d,
$\neq 0$ λ is not adjusted and presumably the user has a computational strategy for λ,

d_2 $\neq 0$ **sub**d_2 search subroutine called,

d_3 $\neq 0$ **sub**d_3 search subroutine called,

d_4 $\neq 0$ **sub**d_4 search subroutine called.

A simple specification is usually sufficient. The following example results in a call to sub2 at convenient stages of the calculation. For this subroutine default values for the other parameters are usually satisfactory.

```
search (-0.02)
```

Notes:

(1) If s=+ve, extrapolation of the fission source occurs concurrently with λ changes and hence this strategy should only be used when the data changes are minimal - for example, with a poison search (Section 6.18.2).

If s=-ve, extrapolation of the fission source does not occur unless $| k - k_{reqd} | \leq k_{acc}'$ and hence this strategy is suitable when data changes are drastic - for example with a critical size search (Section 6.18.4).

(2) To calculate the effective multiplication constant after a search calculation the **search** must be turned off with the data search(0).

In addition to **search** data all the currently available routines require extra data to be entered in the user's array. These data are detailed for each routine later.

wsea (w_k , k=1,2,... to a maximum of *maxw*)

and sufficient storage must be assigned for this array (default maxw=400).

Additional infrequently required search data associated with eigenvalue parameters are,

fsea λ_L, λ_1, λ_2, λ_U, p

where λ_L is a lower limit to λ such that $\lambda < \lambda_L$ will cause program termination (default value 1.-2)

λ_1 is a first estimate of λ (default value 1.)

λ_2 is a second estimate of λ (default value 1.1); if possible it should be associated with a less reactive system ($k_2 < k_1$), since if λ_1 results in $k_1 < k_{reqd}$ then $\lambda_2^* = \lambda_1 - (\lambda_2 - \lambda_1)$ is used instead of λ_2 to hopefully increase k,

λ_U is an upper limit to λ such that λ_U will cause program termination (default value 1.+2), and

p is an extrapolation limit parameter (default value 3.); if λ_{n-1} and λ_n are consecutive estimates of λ, the next estimate is λ_{n+1} and m is an index such that

$$| \lambda_{n+1} - \lambda_m | = \min (| \lambda_{n+1} - \lambda_n | , | \lambda_{n+1} - \lambda_{n-1} |)$$

then if $| \lambda_{n+1} - \lambda_m | < p | \lambda_n - \lambda_{n-1} |$

we accept λ_{n+1}, otherwise we take

$$\lambda_{n+1}^* = \lambda_m + p \ \text{sign} (\lambda_{n+1} - \lambda_m) | \lambda_n - \lambda_{n-1} | \ .$$

The simple chord method is used to estimate a critical value of λ because during early stages of the calculation the values of k are inaccurate, a result of only partially converged fluxes. More complicated interpolation or extrapolation is unlikely to be better and may well be worse. The hesitant extrapolation process would be desirable no matter what method were used for estimating λ_{n+1}. The process does permit extensive extrapolations and after several extrapolations, the estimate could be:

$$\lambda_{n+1}^* = \lambda_2 + p (1 + p + p^2 + \ldots + p^{n-2}) | \lambda_1 - \lambda_2 |$$

For example if p=3 :

$$\lambda_5^* = \lambda_2 + 39 | \lambda_1 - \lambda_2 |$$

Hence values of λ_1 and λ_2 which are too close together are not desirable unless they are good estimates of the critical value of λ.

Finally minimum limits on the number of iterations may be set as follows.

lsea l_1 , l_2

where l_1 is the minimum number of outer iterations between consecutive estimates of λ (default value 4), and

l_2 is the number of estimations of λ before fresh inner relaxation factors (ω_g, g =1,2,...,ng) are sought when the solution method is SLOR (default value 4).

6.18.2 sub1 - poison concentration adjustment

The search subroutine **sub1** may be used to adjust the concentrations in selected parts of the reactor of poison materials which have absorption only (set with the **abs** feature - Section 6.14). A poison concentration search may be set using the data

search (*i*.**01**)

where *i* is the number of the additional reaction for poison absorption, and a search strategy with λ updated every fixed number of outer iterations is satisfactory (Section 6.18.1).

Other data may be provided trailing **search**(*i.01*) if the default values are not satisfactory.

All poison concentrations are initially set with **abs** data and may be adjusted at different rates to achieve criticality. A weight w_k must be supplied for each material as follows.

wsea $w_1, w_2, \ldots, w_{maxm}$

If c_k denotes the initial concentration of poison associated with **m**(k) then the adjusted concentration $c_k(\lambda)$, corresponding to an eigenvalue λ, is given by,

$$c_k(\lambda) = c_k \{1 + (\lambda - 1)w_k\}, \; k = 1, 2, \ldots, \text{maxm} \quad .$$

Consider a search on the concentration of poison required in the fuel of the moata3d sample run (Section 6.2) to achieve criticality ($k_{reqd} = 1$). For this a 1/v poison needs to be introduced as, say **m**(5), and **maxm** must be set to at least 5 in the **prelude**. If the additional reaction 3 is used by the **abs** feature (Section 6.14), then **maxs** must be set to at least 3 in the **prelude**.

Example:

```
prelude maxx=23,maxy=17,maxz=17,maxg=4,maxm=5,maxs=3 end
xsd  fuel  m(1) ...
xsd  inv455b m(5)  4*1.-10, 5.43-2,1.42+1,1.69+2,4.18+2, 20*0
abs(3)= 1(1)4  4*5  1.-5,3*0
 ...
search (3.01)  #3 -1.
wsea 1 4*0
start
```

The search statement above uses a SKAN data skipping feature (#3 - Appendix B) and is equivalent to

```
search (3.01)  1.  1.-3  1.-2  -1.
```

However in the example default values for the trailing numbers would suffice, *e.g.*

```
search (3.01)
```

Note: After a search, following a **start** or **restart**, the concentration of the poison is built into the corresponding cross-section in additional reaction *i*. The poison is permanently associated with σ_{rem} of the material (Section 6.14) if the **start** (or **restart**) keyword is followed by the data:

abs (*-i*)

6.18.3 sub1 - general material concentration adjustment

A general search on material concentration (including fissionable materials) is set using the data:

search (-0.01)

wsea w_1 , w_2 , . . . , w_{maxm}

The material concentrations (not just particular reaction concentrations) are then adjusted, using the same rules as for poison concentrations (Section 6.18.2) but with $c_k = 1$, as follows:

$$c_k(\lambda) = \{1 + (\lambda - 1)w_k\} \quad , \quad k = 1, 2, \ldots, \text{maxm} \quad .$$

For example with the data

```
search (-0.01)
wsea   1,4*0
```

and a critical eigenvalue of, say, λ_c, all cross-sections of material 1 would be multiplied by λ_c and all diffusion coefficients of material 1 would be divided by λ_c at the termination of the calculation.

In addition to the above, concentrations may be adjusted of materials that are subsequently mixed to form material **m**(*k*) constituting part of the reactor. Firstly, material **m**(*k*) must be defined (so that it may be included in the reactor layout) by using, say, a null definition statement such as,

defn *matname* **m**(*k*)

Next, special definition records, with the mixing rule to be used during the search calculation, must be entered in the **wsea** array (elements w_{maxm+1}, w_{maxm+2}, . . . to a maximum of w_{maxw}, with one element for each material number given with **defn**). The data takes the form

defn *matname* **wsea** **m**(*k*)= **m**(m_1) $conc_1$ **m**(m_2) $conc_2$ · · ·

where **wsea** is the only difference from a 'normal' definition (Section 6.12).

For a search on the critical ^{235}U enrichment of an essentially natural uranium reactor the following data could be used. Note that the microscopic cross-section changes resulting from, for example, different resonance shielding would be ignored.

```
xsd u235 m(1)...                          - microscopic data
xsd u238 m(2)...                          - microscopic data
xsd mod m(3)...                           - microscopic data
defn uranium m(4)                         - null definition to enable use of m(4) with reg record
defn addu m(5) = m(1) 1 m(2) -1
xm...   ym...   zm...
reg mx...  my...  mz...  m(4)
 ...
search (-0.01)
wsea  4*0,1,0,...,                        - maxm elements must be supplied
defn uranium wsea m(4) = m(2) 1 m(5) 0.007 m(4) 4.78-2
```

The composition of uranium during the actual criticality search calculation is then

$$c_4(\lambda) = \{(1 - 0.007\lambda)\,^{238}\text{U} + 0.007\lambda\,^{235}\text{U}\}\,0.0478$$

For the above example, because **defn** adds 7 numbers to the **wsea** array, *maxw* must be set in the **prelude** so that *maxw* ≥ *maxm* + 7.

Note: At the termination of the criticality search (after **start** or **restart**) the critical value of λ is built into the corresponding cross-sections.

6.18.4 sub2 - mesh width adjustment

The search subroutine **sub2** adjusts selected mesh widths in a reactor and is called with the data,

```
search(-0.02)
```

Data must also be supplied in the array **wsea** which are used to adjust each mesh interval (including boundary conditions) as required. Given the mesh structure below

xm (or **rm**)	$d_L,\ \delta x_1,\ \delta x_2, \ldots,\ \delta x_{nxmi}\ ,\ d_R$	- *nxmi* = number of x mesh intervals,
ym (or **zm**)	$d_I,\ \delta y_1, \delta y_2, \ldots,\ \delta y_{nymi},\ d_O$	- *nymi* = number of y mesh intervals,
zm	$d_B,\ \delta z_1, \delta z_2, \ldots,\ \delta z_{nzmi},\ d_T$	- *nzmi* = number of z mesh intervals,

the following additional data must be supplied

wsea	$w_L,\ w_1,\ w_2, \ldots,\ w_{nxmi},\ w_R, \ldots,$	- *maxx*+1 elements must be supplied,
	$w'_I,\ w'_1,\ w'_2, \ldots,\ w'_{nymi},\ w'_O, \ldots,$	- *maxy*+1 elements must be supplied,
	$w''_B,\ w''_1,\ w''_2, \ldots,\ w''_{nzmi},\ w''_T$	- *nzmi*+2 elements must be supplied,

then the mesh intervals corresponding to an eigenvalue λ are:

$$
\begin{aligned}
d_L(\lambda) &= d_L(1+(\lambda-1)w_L)\\
\delta x_1(\lambda) &= \delta x_1(1+(\lambda-1)w_1)\\
&\cdots\\
\delta x_{nxmi}(\lambda) &= \delta x_{nxmi}(1+(\lambda-1)w_{nxmi})\\
d_R(\lambda) &= d_R(1+(\lambda-1)w_R)\\
d_I(\lambda) &= d_I(1+(\lambda-1)w'_I)\\
\delta y_1(\lambda) &= \delta y_1(1+(\lambda-1)w'_1)\\
&\cdots\\
\mathrm{dy}_{\mathrm{nymi}}(\lambda) &= \delta y_{nymi}(1+(\lambda-1)w'_{nymi})\\
d_O(\lambda) &= d_O(1+(\lambda-1)w'_O)
\end{aligned}
$$

and similarly for the z−components.

For the moata3d sample run (Section 6.2) consider an adjustment to be made to the separation of the fuel tanks to achieve a multiplication constant of 1. This adjustment is not small as the tanks are only loosely coupled. In general large adjustments may need to be anticipated with initially more mesh points than would be needed for an eigenvalue calculation. In this case the adjustment is achieved by changing the extent of graphite, material **m**(3), between the tanks (in the x-direction), designated bold in the following data.

```
prelude maxx=23 maxy=17 maxz=17 ...   end
 ...
wsea  0, 5*1, 17*0, 0   0,16*0,0   0,16*0,0
```

In the following 2D example, the height of the core (**bold data**) is altered to achieve criticality with the reflector (the remaining data) adjusted so that the overall height does not change.

```
prelude maxx=20 maxy=13 ... end
 ...
rm   0.0 5*1.3 10*1.1 0.71
zm   0.0  4*1.3, 8*1.1 0.71
wsea  21*0.,  0.0 4*1. 8*-0.590909 0.0
```

The number of **wsea** entries in the example corresponds to **maxx**=20, although **maxx**=16 would be satisfactory. The mesh intervals are adjusted as follows:

$$\delta z_1(\lambda) = \cdots = \delta z_4(\lambda) = 1.3 + 1.3\ (\lambda-1)$$

$$\delta z_5(\lambda) = \cdots = \delta z_{12}(\lambda) = 1.1 - 0.65\ (\lambda-1)$$

therefore core height $= \delta z_1(\lambda) + \delta z_2(\lambda) + \cdots + \delta z_4(\lambda) = 5.2 + 5.2(\lambda-1)$
and the reflector height $= \delta z_5(\lambda) + \delta z_6(\lambda) + \cdots + \delta z_{12}(\lambda) = 8.8 - 5.2(\lambda-1)$

In this case we recommend that an upper limit to λ be set which ensures that the reflector does not vanish. (Any non-boundary mesh spacing, which is less than 10^{-4}, is set equal to 10^{-4} by POW3D.) This may be done with the following additional data:

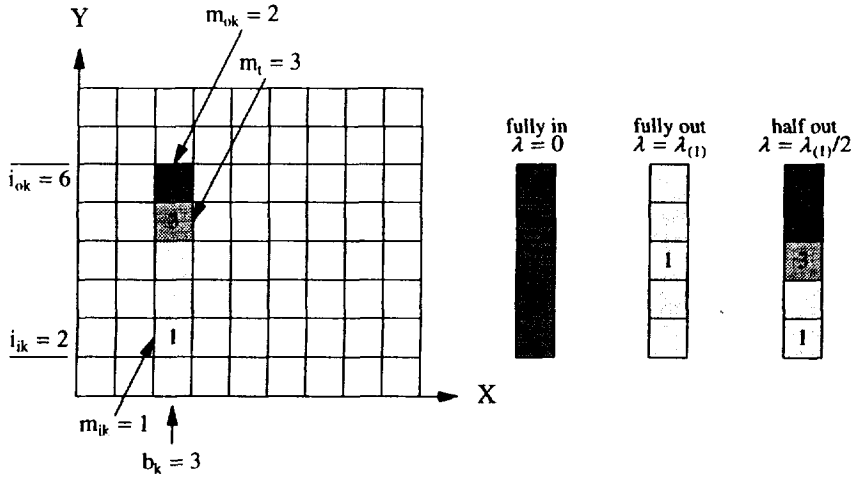
```
fsea   1.-2, 1., 1.1, 2.69, 3.0
```

Note: At the termination of the criticality search, after **start** or **restart** the critical value of λ is built into the corresponding mesh widths.

6.18.5 sub3 - control absorber channel movement

The search subroutine **sub3** is used to adjust the movement of control materials along directions (channels) parallel to an axis. A collection of channels with materials which move together will be called a bank, b_k. The movement of banks forms the basis of the criticality search available here. A bank b_k may be moved from a fully in position corresponding to mesh interval i_{ik} to a fully out position corresponding to mesh interval i_{ok}. All banks start fully in. When one bank b_k is moved to the fully out position, the next bank b_{k+1} may be moved. As a bank moves out, material m_{ik} comes into the channels at mesh interval i_{ik}, while material m_{ok} goes out of the channels at mesh interval i_{ok}. Fine adjustment is carried out by providing a 'tip' material m_t, between the materials m_{ok} and m_{ik}. The composition of the tip is a homogenised mixture of materials m_{ok} and m_{ik} which is varied to correspond to positions of the bank partway between mesh intervals.

Simple examples follow to illustrate the data requirements. The first example consists of only one bank and that of only one channel in a 2D reactor with the following layout.



As well as the normal **reg** data (Section 6.15.2) describing the layout of the reactor, additional data need to be supplied with the array **wsea**. The specification of material filling the control banks is unnecessary in the reactor layout. For the example the following data would be used:

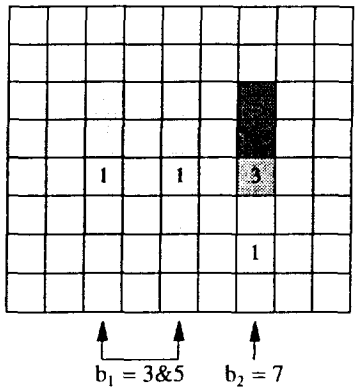
```
search (-0.03)
fsea 0., 0.4, 0.6, 1., 2.
```

```
wsea    3,      -1     =1.      2       2,   6    1,   2    3,1
          ↑       ↑      ↑       ↑       ↑    ↑    ↑    ↑    channel
        tip material  bank 1  λ(1)   moves // to y=axis  i_i1 i_o1 m_i1 m_o1 position
                                                                          x,z
                                    (1= moves // to x-axis)               (y,z)
                                    (3= moves // to z-axis)               (x,y)
```

To associate more than one channel with a bank, the channel positions must be specified successively. For example, materials filling intervals 3 and 5 may be moved together using the data:

wsea 3 **-1=1. 2 2,6 1,2 3,1 5,1**

To specify more than one bank, the equivalent of the bold data in the statement above must be repeated for each bank in turn. An example with two banks is illustrated below.



Data for the example follow:

```
wsea  3  -1=2.  2  2,6  1,2  3,1  5,1  -2=1.  2  2,6  1,2  7,1
```

The eigenvalue λ is associated with the banks in the following ways:

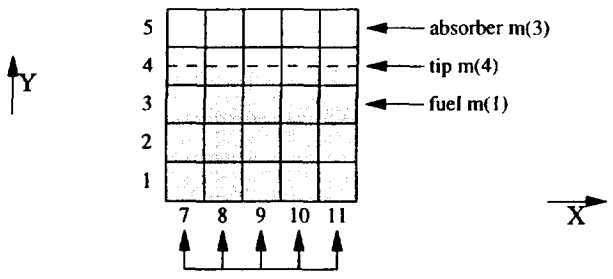
$\lambda = 0$ - all banks in,
$\lambda = |\lambda_{(1)}|$ - bank 1 out, all other banks in,
$\lambda = |\lambda_{(1)}| + |\lambda_{(2)}|$ - banks 1 and 2 out, all other banks in, etc.,

with fractional interpretation as explained earlier. Since this correspondence requires reactivity increase or decrease to be continuously associated with increasing λ , *in* and *out* may need to be interchanged in the definition of bank b_k. This may be achieved by setting the λ-limit for a bank negative. If the data for the example is changed to

```
wsea  3  -1=-2.  2  2,6  1,2  3,1  5,1  -2=1.  2  2,6  1,2  7,1
```

then fully in means all control channels filled with material 1. The choice $\lambda_{(1)}$ etc. is somewhat arbitrary, except for sign, but an attempt should be made to maintain a smooth change in reactivity as a new bank begins to move out.

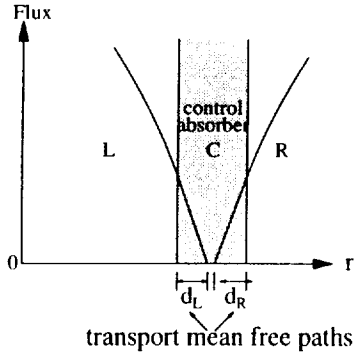
As an example of an application of **sub3**, consider a 2D version of the moata3d calculation of Section 6.2. Suppose that the fuel loading to give k_{eff}=0.9 is required. One way to determine this is by moving 5 fuel channels together as 1 bank.



The data needed for this example, additional to the moata3d data given earlier, are listed below.

```
prelude  ...  maxm=4  end
search(-0.03)  0.9
wsea  4  -1=1.25  2  1,5  1,3  7,1  8,1  9,1  10,1  11,1
```

Note: Diffusion theory fails for control regions of 'strong' absorbers and transport corrections need to be applied to the data. Only 1D approaches will be considered. One method available (G. Doherty, AAEC unpublished report) is to adjust the diffusion coefficient of the absorber in a diffusion calculation so that the same flux is obtained as for a collision probability transport calculation. An alternative less desirable method is based on the extrapolated boundary concept illustrated below.



We have

$$D_L \frac{\partial \phi}{\partial r}|_L + \frac{\phi_L}{3d_L} = 0 \text{ and } -D_R \frac{\partial \phi}{\partial r}|_R + \frac{\phi_R}{3d_R} = 0 \ .$$

Since internal boundary conditions of the preceding type are not permitted with POW3D the conditions must be simulated using modified data for the control absorber, C,

$D_C = 0$ (10^{-4} is used instead to prevent code error)

and

$$\sigma_{rem,C} = \frac{2}{\bar{l}} \left(\frac{1}{d_L} + \frac{1}{d_R}\right) \ ,$$

where $\bar{l}$ = mean chord length for C (=4V/S).

The adjustment may be made to the data of C for some, or all, of the groups.

6.19 Fixed External Source Specification

A fixed source about points (in boxes around the intersection of mesh lines), for all groups, may be specified using **fsce** followed by *maxx*×*maxy*×*maxz*×*maxg* data entries. Storage for the source is provided by setting *maxf*=1 in the **prelude**. The source array is 'dimensioned' as if it were fsce(maxx,maxy,maxz,maxg,maxf) - using virtual input/output (Appendix C3) - and the data to fill it must be ordered by x, y, z directions and by group, as follows:

fsce

s(1,1,1,1,1), s(2,1,1,1,1),...,s(maxx,1,1,1,1),

s(1,2,1,1,1), s(2,2,1,1,1),...,s(maxx,2,1,1,1),...

s(1,maxy,1,1,1), s(2,maxy,1,1,1),...,s(maxx,maxy,1,1,1),

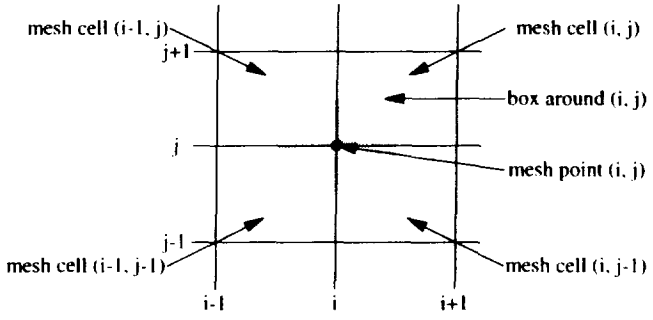
s(1,1,2,1,1), s(2,1,2,1,1),...,s(maxx,1,2,1,1),...

s(1,maxy,maxz,1,1), s(2,maxy,maxz,1,1),...,s(maxx,maxy,maxz,1,1),...

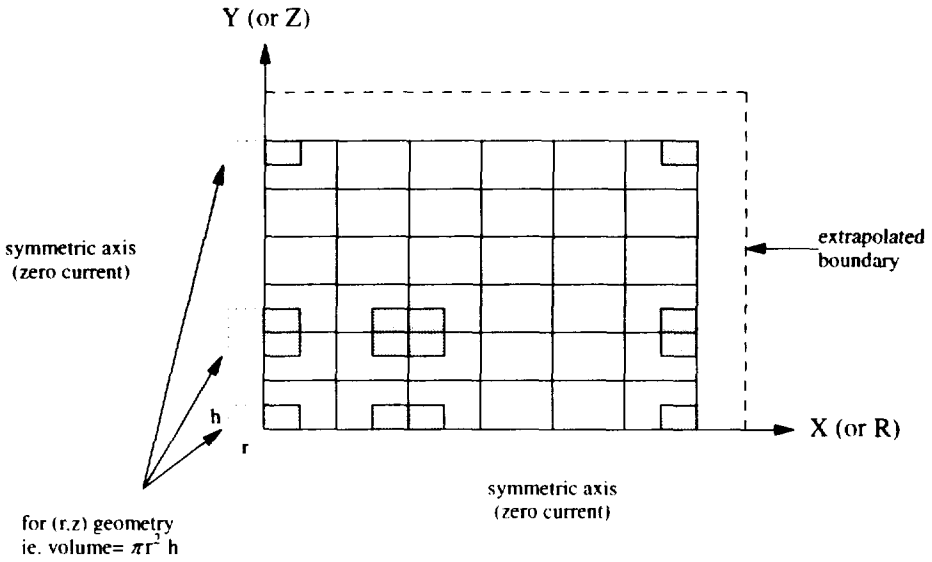
s(1,1,1,2,1), s(2,1,1,2,1),...,s(maxx,1,1,2,1),...

s(1,maxy,maxz,maxg,1), s(2,maxy,maxz,maxg,1),...,s(maxx,maxy,maxz,maxg,1).

The source s(i,j,k,g,l), in units of n sec^{-1} (or appropriately scaled for ease of presentation), is the total fixed source for group g entering a box around the mesh point (i,j,k) as illustrated (for 2D). The box extends midway to the neighbouring mesh points.



Note that along boundaries the code puts the source into fractions of boxes as in the illustration below. Hence the user needs to enter the appropriate source for the fraction of the box. For XY geometry if the source is entered along a reflective boundary the fraction is 0.5 and at the intersection of two reflective boundaries, the fraction is 0.25. However for RZ geometry the 'boxes' are actually annular regions hence the source along the z axis is entered into volumes of $\pi r^2 h$ and there is no such complication except at a reflective boundary along the r axis.



For spherical geometry the central box has the volume $4/3\pi r^3$.

In order to simplify source data presentation, the following feature using an argument range, is available. Note that the first actual source data must be given as real data (with a decimal point and/or exponent).

fscel (1-3,1,1,1)=s(1,1,1,1,1), s(2,1,1,1,1), s(3,1,1,1,1) - for the x-direction, or
fscel (1,1-3,1,1)=s(1,1,1,1,1), s(1,2,1,1,1), s(1,3,1,1,1) - for the y-direction, or
fscel (1,1,1-3,1)=s(1,1,1,1,1), s(1,1,2,1,1), s(1,1,3,1,1) - for the z-direction, or
fscel (1,1,1,1-3)=s(1,1,1,1,1), s(1,1,1,2,1), s(1,1,1,3,1) - for groups, or
fscel (1,1,1-3) equivalent to above (z omitted) - for compatibility with POW, or
fscel (1-2,1-2,1,1)=s(1,1,1,1,1), s(2,1,1,1,1), s(1,2,1,1,1), s(2,2,1,1,1)

Consider an example of a unit line source (1 n cm^{-1} sec^{-1}), in the fastest energy group, centred at the origin of an xy system with reflective x and y boundaries. The following data could be used:

```
fsce  0.25  100000*0
```

The value 0.25 is used because the source is entered into a 1/4 box. Too much data are of no consequence. If this calculation is the first to use the **fsce** array then the following data would suffice.

```
fsce  0.25
```

As a further example if a unit centre point source is required (1 n sec^{-1}) in a reflective rz system, with strengths of 0.7, 0.2, 0.1, 0 in groups 1, 2, 3 and 4 respectively then the following data could be used:

```
prelude maxx=25,maxy=27,maxg=4,maxf=1 end
fsce  0.35, 674*0., 0.1, 674*0.,  0.05,100000*0
```

or more simply (and if the array first needs to be cleared),

```
fsce  100000*0
fscel(1,1,1,1-3) = 0.35, 0.1, 0.05
```

or since the third, z, dimension may be omitted for compatibility with POW data,

```
fscel(1,1,1-3) = 0.35, 0.1, 0.05
```

The source may also be entered as a source density by following the **fsce** data with the keyword,

fsceconv

which converts the supplied data to the required volume integral.

Note: The fission emission reaction is used to assess the convergence of a calculation which includes fission (Section 6.22.1). In a source calculation, if fissionable material is not present, convergence is normally based on the removal reaction but may be changed to POW3D reaction number *n* with:

nofiss = *n*,

The POW3D reaction numbers are listed below (see also Appendix A2). The additional reaction numbers depend on the number of POW3D groups *ng*.

1	- σ_{tr}	
2	- σ_{rem}	default reaction for assessment of convergence if there is no fission
3	- $\nu\sigma_f$	usual reaction for assessment of convergence with fission
4+*ng*	- σ_{r1}	first additional reaction, default σ_p
5+*ng*	- σ_{r2}	second additional reaction, default σ_{tot}
6+*ng*	- σ_{r3}	third additional reaction, default σ_f
...		

Consider a 4 group calculation (ng=4) in which the first additional reaction is set to absorption using the **abs** option (Section 6.14). To obtain convergence criteria based on the total absorption the following data could be used.

```
nofiss=8
```

6.20 Trial Flux Specification

For most runs a user would not be concerned with the default trial flux generated by POW3D because it is always adequate. (POW3D uses a flux appropriate to a bare homogeneous system as a trial solution.) However the user should be aware of several points.

(1) A trial flux may be used from a flux dump of a similar job (**restart** - Section 6.24).

(2) Except for the first calculation of a run the flux solution of the previous calculation is used unless an element of flux is zero (as it would be if there were more mesh points now than before).

(3) A POW3D generated trial solution may usually be forced by entering the data,

```
flx=0
```

but if `calc=`**adjoint** the following is needed:

```
flxa=0 .
```

6.21 Coarse Mesh for Region Rebalance

Much of the calculation strategy employed by POW3D is set with data read from disk (unit 4) as part of the normal use of SKAN routines (Appendix B). The strategy is briefly discussed in Section 3 and although it may be varied the user would infrequently, if ever, need to make changes. Over the past years the strategy data has been 'tuned' for a variety of calculations and this 'tuning' is likely to continue. The only data the user may wish to change (although this is unlikely) is the coarse mesh associated with region rebalance. (Extensive tuning is considered by Barry and Pollard, 1987 revised 1995.)

For region rebalance, coarse mesh segmenting boundary lines of the solution plane may be specified by the user prior to each **start** (or **restart**) as follows:

lx mx $i_1, i_2, \ldots, i_{mx+1}$
ly my $j_1, j_2, \ldots, j_{my+1}$
lz mz $k_1, k_2, \ldots, k_{mz+1}$

where mx is the number of coarse mesh segments in the x (or r) direction,

and $i_m \le i < i_{m+1}$ identifies the lines i forming the m^{th} coarse mesh segment in the x direction, with $i_1 = 1$ and $i_{mx+1} = nxm$, (nxm is the number of x-mesh solution lines). Similar meanings apply in the other directions for j and k.

The user must also set $maxr$ in the **prelude** so that $mx \times my \times mz \le maxr$. The default value for $maxr$ is the product $\sqrt{maxx}\sqrt{maxy}\sqrt{maxz}$. Storage allocated for the solution array is $maxr \times (maxr + 1)$ elements.

The choice of plane segmentation is not always clear but a good rule (in the absence of specific experience) is to attempt to isolate regions which may be slow to converge (for example fuel regions or strongly diffusing regions in the vicinity of fuel regions). The minimum coarse mesh data which may be specified by the user is $maxr$ in the **prelude**. In the absence of further data a simple (but usually effective) procedure is adopted by the code to generate mx, my, mz as well as the segmenting boundary lines.

As a simple example consider isolating the fuel region in the moata3d calculation of Section 6.2. The default procedure would do this anyway. Nevertheless the following data could be used:

```
prelude maxx=23,maxy=17,maxz=17,...,maxr=12 end
lx=3 1,7,12,23
ly=2 1,6,17
lz=2 1,7,17
start
```

6.22 Calculation Termination Conditions

The default termination conditions for a POW3D calculation should usually be adequate. A calculation ceases when any one of the following termination conditions is met:

(1) the calculation converges within the set accuracy limits:

  (i) overall neutron balance accuracy $\le acclam(1)$, default 1.-4,

  (ii) local fission source accuracy $\le accfo(1)$, default 1.-4, and

  (iii) if a search is requested (Section 6.18), k effective accuracy $\le k_{acc}$ (default 1.-3);

(2) machine time exceeds a specified limit (default 90% of the time remaining for current job step),

(3) number of outer iterations, n, reaches a set limit (default `nol=100`),

(4) *nil*, the set limit to the number of inner iterations, is 0.

If a previous flux dump is entered for edit purposes using the **restart** option (Section 6.24), further calculation on it may be bypassed by setting a zero limit on the number of inner iterations (Section 6.22.4).

If an input data error is encountered, POW3D does not begin a calculation but simply 'browses' through the remaining data.

6.22.1 Accuracy termination (acclam and accfo)

POW3D uses two main quantities, which change with each outer iteration n, to assess convergence -

(i) overall neutron balance, $b^{(n)}$, and

(ii) local reaction (usually fission source) error, $S_{err}^{(n)}$.

In addition for a criticality search, a further quantity needs to be considered,

(iii) effective multiplication constant error, $| k^{(n)} - k_{reqd} |$.

Let $S_{ijk}^{(n)}$ be the total reaction of indicated type for outer iteration number n in a box about the mesh point (i,j,k), then $S_{ijk}^{(n)}$ has the following meanings:

calc=real, eigenv, $$S_{ijk}^{(n)} = \sum_g \phi_{ijkg}^{(n)} \int_{i,j,k} \nu\sigma_{f,g}\, dV \ ;$$

calc=real, source, same as above if any reactor material is fissile,

otherwise $$S_{ijk}^{(n)} = \sum_g \phi_{ijkg}^{(n)} \int_{i,j} \sigma_{nofiss,g}\, dV \ ,$$

and *nofiss* is usually 2 (removals) as indicated in Section 6.19;

calc=adjoint, $$S_{ijk}^{(n)} = \sum_g \phi_{ijkg}^{(n)*} \chi_{pg} \ .$$

Then the overall neutron balance is,

$$b^{(n)} = \sum_i \sum_j \sum_k S_{ijk}^{(n)} \ / \ \sum_i \sum_j \sum_k S_{ijk}^{(n-1)}$$

and the local reaction error is,

$$S_{err}^{(n)} = \left\{ \max_{i,j,k} \left[S_{ijk}^{(n)} / S_{ijk}^{(n-1)} \right] - \min_{i,j,k} \left[S_{ijk}^{(n)} / S_{ijk}^{(n-1)} \right] \right\} S^{(n-1)} / S^{(n)}$$

where (i,j,k) are all mesh points for which $S_{ijk}^{(n-1)} \neq 0$ and $S^{(n-1)}$ and $S^{(n)}$ are total source values.

POW3D utilises a convergence count down scheme from trial solution (stage 3) to final convergence (stage 0). The stages for the usual problem are summarised below.

Stage	Procedure	Condition for change to next stage
3	Power iteration	$b^{(n)} \leq$ acclam(3) and $S_{err}^{(n)} \leq$ accfo(3)
2	Chebyshev extrapolation, $4 \leq$ order ≤ 6	$b^{(n)} \leq$ acclam(2) and $S_{err}^{(n)} \leq$ accfo(2)
1	As above but more confident, $6 \leq$ order ≤ 10	$b^{(n)} \leq$ acclam(1) and $S_{err}^{(n)} \leq$ accfo(1)
0		converged (provided $| k^{(n)} - k_{reqd} | \leq k_{acc}$ for a search calculation - Section 6.18)

The default values for the accuracy limits are:

```
acclam(1) = 1.-4   acclam(2)= 1.-3   acclam(3)= 1.-2
accfo(1)  = 1.-4   accfo(2) = 1.-3   accfo(3) = 1.-2
```

Usually only final accuracy limits, *acclam*(1) and *accfo*(1), are of interest and may be changed as follows:

```
acclam = 1.-3
accfo = 1.-3
```

Note that the tight limits, `acclam=1.-5, accfo=1.-5` are seldom, if ever, required.

6.22.2 Machine time limit (tlim)

The time limit parameter is used to ensure that if time is exceeded, the calculation is terminated by POW3D with output rather than abruptly by the operating system. The time limit may be set with the data:

tlim *f*

where *f* is the fraction of remaining (cpu) time for the job step (set with AUS invocation - Section 4) beyond which termination of the calculation (and output) is forced.
(Note that *f* is converted and stored as machine time limit in minutes .)

The job does not terminate completely but following the **start** (or **restart**) data the keyword **tlim** is sought. The job will terminate, however, with the keywords **end** or **stop**. After resetting the time limit (default `tlim = 0.9`) POW3D proceeds with editing requirements before program termination.

Example:

```
tlim = 0.95
start
tlim = 0.95
```

6.22.3 Outer iteration count limit (nol)

The current calculation terminates if the number of outer iterations reaches the set limit **nol** (default value 100). The limit may be changed, for example, with the data:

```
nol = 10
```

6.22.4 Inner iteration count limit (nil)

If the inner iteration count limit **nil** is exceeded for one energy group, the next group is started in the normal solution process (default `nil=100`).

A **restart** may be used to edit a previous flux dump (Section 6.23) by setting the limit to zero:

```
nil = 0
```

For further calculations the limit must be reset to the required value *e.g.*

```
nil = 100
```

6.23 Output Requirement Specification

Much of POW3D output is optional and may be selected with the following keywords:

output [data source fl1 flux flux3d fl3 print printe dataft uedit pdump vis cram gog]

The above descriptive keywords which follow **output** are optional and have the following meanings,

data - print cross-section data (before the calculation is started),

source - print (edge) source for a source calculation,

fl1 - for 2D calculations, on termination produces a POW-type flux dump at the end of the file on unit 24 which may be used by other AUS modules, or as a **restart** file by POW3D,
- the same file may be used for many different jobs,

flux - equivalent to fl1 except that the file is written prior to uedit call,

flux3d - POW-type flux dump extended to 3D, for use by other AUS modules,
- POW3D cannot **restart** from this type of flux dump,

fl3 - at the termination of a calculation produces a POW3D-type flux file on unit 61 which functions as a **restart** file (see Appendix C),
- a separate file should be used for each distinct job,

print - print (edge) flux (integer values which are multiplied by the printed scaling factor) *e.g.* printed flux of 1102877 and scaling factor of 1.e+08 means that $\phi = 1.102877 \times 10^{-2}$),

printe - (edge) flux is printed with exponents but with the decimal point omitted after first digit *e.g.* 1103-2 means that $\phi = 1.103 \times 10^{-2}$,

dataft - print cross-section data, layout and mesh spacing after a search calculation,

uedit - calls subroutine **uedit** which may be supplied by the user (Section 8.2),
- the default **uedit** normalises flux (for **printe**, **pdump**, **edit**, **edits** but not **print** or **flux**) to a specified power (for a segment of the reactor when reflective boundaries are used) *e.g.*

```
power=1.e+7, 3
```

where power (W) is followed by additional reaction number (=3) for the fission cross-section,

pdump - print a dump of all POW3D variables referenced in the **list** variable [Pollard, 1974, Appendix D4],
- this option was set up to assist program implementation and has been extended for use with user supplied DATRAN **pdump***n* routines (Section 8.1, Datran(A.4)/(H)),

vis - produce geometry and flux information - via a file or directly transferred for 3D scientific visualisation using, say, AVS [1994],
- see edit plotting specification (Section 6.26.3, (iii)) for more information,

gog - produce data for GOG [Hopkins and Oakes 1968] on unit 2 (for a 2D eigenvalue problem only),

cram - produce data for CRAM [Hassitt 1962] on unit 2 (for a 2D eigenvalue problem only).

The **gog**, and **cram**, **ouput** options were set up mainly for comparison tests of the POW3D predecessor, POW, with other codes. They are retained for historical reasons.

The default **output** option is:

```
output  data  source  print  dataft
```

If the optional output is not required the single keyword is used:

```
output
```

Another output option is available but is not specified with the **output** keyword. This option enables summary output of specific interest to the user to be written and the termination information of the run to be read. The option is invoked as first output after every outer iteration, or kinetics time step, provided a DATRAN coded routine **summary** (Section 8.1, Datran(A.4)/(F)) is supplied by the user.

Output may be produced at intermediate stages of the calculation for items usually only produced on termination of the calculation. For this option, the keyword **iout** is followed by one number for each **output** item, in the same order as the output items. For a steady state calculation **iout** refers to every outer iteration. For kinetics **iout** refers to every time step. For example the data

```
output  data  flux  print
iout  0  0  5
```

will result in the following output:

data - at the start of the calculation,
flux - at the termination of the calculation (created at the end of the file), and
print - every 5th outer iterations and also at the termination of the calculation.

The default values are:

```
iout  0  0  0
```

6.24 Calculation Start

A calculation is started when the following keyword is encountered

start

or if a flux dump is to be used as a trial flux

restart

The flux dump specified is only used if several important variables, including **calc** (Section 6.17), match those for the current job. If an acceptable dump is not located, a **start** is initiated. A separate POW3D-type restart file, **fl3** (on unit 61), should be used for each distinct calculation.

For consecutive calculations the following options allow the use of some previously calculated data:

restart	flux, k, critical eigenvalue and iterative parameters *etc.* are retained,
restart continue	flux, k and critical eigenvalue only are retained,
start	flux only retained unless some element of flux is zero and a trial flux is then created (Section 6.20),
flx=0 start	data from a previous calculation is not used and a trial flux is created (Section 6.20).

Restart is normally from a POW3D-type flux dump (**fl3** on unit 61). A suitable dump needs to be selected from the dumps in one file using **info** data in **prelude**. Appendix C1 provides further details. Although not the normal requirement, restart from a POW-type flux dump (**fl1** on unit 24) is supported and is briefly covered in Appendix C2. Note that the 3D POW-type flux dump (**flux3d**) is not supported for restart.

6.25 Region edit (first word + ve)

The region edit option produces cross-sections and/or reaction rates which are (real) flux weighted over regions of the reactor. Output may be directed to the printer (unit 6), or as formatted records to disk (*e.g.* unit 12). Cross-sections may be stored in memory for later use or for the creation of an AUS library. The following data are used with this option:

groups, collapsed group structure,

mreg, material regions,

mxs, cross-section and material region correspondence,

edit, initiates the edit calculation, and

write lib, if an AUS cross-section file (Appendix A) is to be produced.

6.25.1 groups, collapsed group structure

The required collapsed group structure is specified with the data:

groups $n, g_1,\ldots,g_{n+1}$

where n is the number of collapsed groups,
g_1 is the first library group of collapsed group 1, and
g_i for i > 1, is the last library group of collapsed group i-1.

For example, if all groups of a 26-group set were required, then the following data could be used:

```
groups 26,1,1(1)26
```

6.25.2 mreg, material regions

Regions required for **edit** output are identified by the numbers of the materials in those regions as follows:

mreg $m_1, m_2, m_3, \ldots$

where m
- +ve represents the material region number, namely the region filled with material m (defined by **reg** and **m**(m) data Section 6.15.2),
- -ve then $|m|$ is the union of all material regions following the negative number up to either the end of the list or a non positive number (0 is otherwise ignored).

Single material regions and unions of regions can be specified in the same statement, for example:

```
mreg 1,2, -3=1,2,3,4, -4=1,3,0, 5
```

Output will be produced for 5 material regions. Output regions 1,2 and 5 are single material regions. Output region 3 is the union of regions 1,2,3 and 4 while output region 4 is the union of regions 1 and 3.

Notes:

(1) Material regions not present in the reactor are ignored. For example, to obtain data for a material appropriate to the whole reactor, all the constituent material regions should be selected as follows:

```
mreg   -1=1(1)30
```

(2) The **reg** data (Section 6.15.2) may need to be modified to select isolated regions of the reactor. If so, the original layout may be saved on disk and readily restored when required (Section 7.16.1).

(3) The **mreg** array is dimensioned *maxn* in the **prelude** (default `maxn=100`).

6.25.3 mxs, cross-section and region correspondence

The cross-sections of materials may be averaged over the flux in regions which do not contain those materials. The required data (if the default values are not satisfactory) follow:

mxs $i_1, i_2, \ldots, i_m, \ldots$

where i is the material number of required cross-section data
m is the material region number over which flux weighted averaging is required.

For example:

```
mreg  2,1,-3=1,2
mxs   5,6
```

will result in output for 3 materials. The first will be averaged over regions filled with **m**(2) using **m**(6) cross-sections. The second will be averaged over regions filled with **m**(1) using **m**(5) cross-sections. The third will be averaged over regions filled with **m**(1) and **m**(2) using **m**(5) and **m**(6) cross-sections respectively. The default value of the vector **mxs** is:

```
mxs=1,2,3,...
```

In the absence of **mxs** data, the output is for the constituent materials of the regions specified with **mreg** data.

6.25.4 edit, initiates the edit calculation

The required **edit** output options are specified with the following data:

edit *l m n p*

where *l* +ve is an identifying number for output,

m -ve the reaction rates are per unit volume,

|*m*| { 1 reaction rates, 2 cross-sections, 3 both,

n { -ve data prepared in memory, +ve unit number for output (6 for printer),

p not used at present.

Cross-section data in memory are replaced (provided n < 0 and m ≥ 2) for all materials requested with **mreg** and **mxs** data. For example, if the default values for **mxs** are used then the data

```
mreg 1,2,-3=4,5
```

results in m(1), m(2) and m(3) being replaced in memory. In the following example,

```
mxs 11,12,13,14,15
mreg  1,2,-3=4,5
```

data for m(11), m(12) and m(13) are replaced rather than m(1), m(2) and m(3). In addition the fission spectrum, **sp**, is collapsed and **ng** is set to the collapsed number of groups when *n* = -2 is used. In this case further editing is not possible and for any subsequent calculation the regeneration of a trial flux solution needs to be forced before the next **start** keyword using the data:

```
flx=0.0
```

To print 4-group reaction rates for each material and also for a mixture of all materials, in the moata3d calculation (Section 6.2), the following data would be used:

```
groups   4  1,1(1)4
mreg     1,2,3,4, -1=1,2,3,4
edit     1,1,6,0
```

The definitions of the various **edit** output quantities, listed below, are simplified by use of an imagined continuous spatial representation, rather than the finite difference form used in POW3D calculations.

(i) reaction rates, for fission emission and the additional reactions

$$RR(M,G) = \sum_{g \epsilon G} \int_M \sigma_g \ \phi_g \ dV$$

(ii) transport reaction rate

$$(TR)RR(M,G) = \left[\sum_{g \epsilon G} \int_M \phi_g \ dV \right]^2 / \sum_{g \epsilon G} \left[\frac{(\int_M \phi_g \ dV)^2}{\int_M \sigma_{tr,g} \ \phi_g \ dV} \right]$$

(iii) removal reaction rate

$$(REM)RR(M,G) = \sum_{g \epsilon G} \int_M \sigma_{rem,g} \ \phi_g \ dV - \sum_{g \epsilon G} \sum_{g' \epsilon G} \int_M \sigma_{s,g \rightarrow g'} \ \phi_g \ dV$$

(iv) scattering reaction rate

$$(S)RR(M, G\rightarrow G') = \sum_{g\epsilon G} \sum_{g'\epsilon G'} \int_M \sigma_{s,g\rightarrow g'}\ \phi_g\ dV \quad \text{for } G \neq G', \text{also}$$

for P_n data, $n\neq 0$ and $G' = G$

$$(S)RR(M, G\rightarrow G) = 0 \text{ (for in memory } P_o \text{ data)}$$

(v) fission spectrum

$$\chi(M, G) = \sum_{g\epsilon G} \chi(g)$$

where χ denotes the normalised fission spectrum ($\sum_g \chi_g = 1$) and the summation is over all groups

(vi) cross-sections (for all reactions)

$$XS(M, G) = RR(M, G) / \sum_{g\epsilon G} \int_M \phi_g\ dV$$

In the above definitions G′ and G on the left represent collapsed group numbers and on the right they represent sets of numbers of groups to be collapsed; M designates a region of the reactor.

6.25.5 write lib, AUS cross-section file creation

As part of the AUS scheme requirement POW3D can both read and write AUS cross-section files. Although files with tabular resonance shielded cross-sections can be read (Section 6.6), only files with shielding for one composition can be written. An AUS file is written using cross-section data held in memory (perhaps prepared with `edit 1 2 -2 0` as described in the previous section). Group energy boundaries, burnup and mass information are not stored by POW3D but can be obtained, if needed, from an *extra* AUS cross-section file. To write an AUS cross-section file the following data are required:

mreg $m_1, m_2, \ldots$
write *name* **lib** *unit* [*extra*]

where	$m_1, m_2, \ldots$	> 0 are the required material numbers (interpreted via the **mxs** array),
	name	is the user's choice of name for the output library, if the *name* is **input** then an AUS cross-section file is not created - see note (4),
	unit	is the FORTRAN unit number of the output library, usually 10,11 or 18 and
	extra	is the FORTRAN unit number of the library from which to get the group energy boundaries, burnup information *etc.* (see note (3)) for the newly created library, or 0 (the default value) if the feature is not required.

Notes:

(1) If data for $P_1, P_2, \ldots$ scattering matrices are held in memory (Section 6.8) then these pseudo materials do not need to be specified with **mreg** data because AUS automatically includes $P_1, P_2, \ldots$ scattering for the material. If $P_1, P_2, \ldots$ data are not wanted then the names of the pseudo materials should be changed from **ditto** to something else.

(2) If averaged group velocities, v_g, are required in the library (*e.g.* for kinetic studies) then:

(i) cross-section data for **inv455b** (Section 6.6.1) must be extracted from a library, and

(ii) if a group collapse is involved, the **inv455b** data should be averaged over the reactor spectrum.

The material number corresponding to **inv455b** should not be included with the **mreg** data unless the **inv455b** cross-section data are also required in the library.

(3) Usually information is obtained for materials from the *extra* file, which have the same names as on the file being created. If materials with the same names are not available then **mxs** $i_1, i_2, \ldots i_m, \ldots$ data (Section 6.25.3) may be used to get burnup and mass information for material **m**(m) from a material in the *extra* file with the same name as **m**(i_m).

(4) For compatibility with the following instruction (Section 6.7)

```
read input lib on 2,0
```

the data

```
write input lib on 2,0
```

is equivalent to

```
edit 99,2  2,0
```

6.26 Point edit (first word -ve)

The point **edit** option produces a map of reaction rates or of real or adjoint fluxes. The reaction rates are calculated at mesh points by averaging over boxes delineated by grid lines midway between mesh points. The keywords used with this option are:

groups, collapsed group structure (Section 6.25.1),

mreg, regions for inclusion in purejoy plot (Section 6.25.2),

mxs, cross-section and region correspondence (Section 6.25.3),

lx, ly, lz, grid specification of required printed output,

plot, must precede each **edit** for which a plot is required,

edit, initiates the edit calculation.

6.26.1 lx, ly, lz grid specification for printed output

Printed output is produced only for the grid points requested. The data requirement is

lx nx $i_1, i_2,...,i_{nx}$
ly ny $j_1, j_2,...,j_{ny}$
lz nz $k_1, k_2,...,k_{nz}$

where nx, ny, nz are the total number of mesh points for output in each of the x, y, z directions,
$i\ j\ k$ specify the required i^{th}, j^{th}, k^{th} mesh points in the x, y, z directions, and
$i_1, i_2,...,i_{nx}$ must be in ascending order, as must the j's and k's.

Note that the grid specification is ignored for plot output.

6.26.2 edit, initiates the edit calculation

The required edit options are specified with the data:

edit $l\ m\ n\ p$

where l -ve is an identifying number for output;

m
- +ve indicates the required additional reaction number,
- 0 specifies absorption (removals - outscatters),
- -ve indicates that reaction -m is required (*e.g.* -3 for fission emission);

n -ve reaction rates are per unit volume (the usual choice);

$|n|$
- 0 flux only (ϕ if p=0, ϕ^* if p=1),
- 1 reaction rates,
- 2 power map (reaction rates with fission energy release, fer_k, weighting),
- 3 reaction rates with weighting factor of $wsea_k$ (the user's array),
- 4 reaction rates with weighting factor of $wsea_{g+(k-1)maxg}$, where k is the material number of the cross-sections used (interpreted through **mxs**), and g the energy group of the original set;

p
- 0 real flux, ϕ, weighting spectrum used,
- 1 adjoint flux, ϕ^*, weighting spectrum used.

6.26.3 plot, plotting specification

Along with a flux or reaction rate print, 1D, 2D and 3D plots of corresponding quantities may be obtained on graphics devices. The plotting requirements (which must precede the **edit** keyword) are set with the data

plot *p string-of-data* 0

where *p* -ve printed output is not produced, only a plot,

| *p* |	0 no plot (default), 1 x plot for given values of y and z, 2 y plot for given values of x and z, 3 z plot for given values of x and y, 4 contour and 2D plot for an xy plane, 5 contour and 2D plot for an xz plane, 6 contour and 2D plot for a yz plane, 7 3D data for a scientific visualisation package;
string-of-data	see line plot option (i) if | *p* | equals 1,2 or 3, see purejoy and contour plots option (ii) if | *p* | equals 4, 5 or 6, see scientific visualisation plot option (iii) if | *p* | equals 7;
0	trailing the *string-of-data* indicates termination of data.

On exit from the point edit in hand POW3D sets $p = 0$. Hence new plot data must be supplied with every plot requirement, or at least the erased *p* must be restored with the data,

```
plot p
```

and the remainder of the previous list is then used. A more detailed description of available options follows.

(i) The line plot option (*i.e.* \| *p* \| equal to 1, 2 or 3)
For the line plot options, the calculated reaction rate or flux function $f_G(x,y,z)$, for collapsed group G, is plotted against x or y or z, while keeping the other two coordinates constant [Trimble 1978]. The *string-of-data* consists of coordinate pairs which select lines parallel to the chosen axis for plotting as follows:

plot *p*, q_1, q_2,...,0

where q_1 , q_2 are
- y,z mesh points for an x plot (\| *p* \| = 1),
- x,z mesh points for a y plot (\| *p* \| = 2),
- x,y mesh points for a z plot (\| *p* \| = 3),

q_3 , q_4 and subsequent optional pairs are defined in the same way.

The following example would result in a group 4 flux plot for 2 lines parallel to the y axis.

```
groups   1   4,4
plot     2     1,1     6,3     0
edit    -1 ,  0 ,  0
```

One plot is along the line through the first mesh point both in the X and Z directions, *i.e.* the Y axis. The second plot is along a line through the 6th x-mesh point and 3rd z-mesh point.

The user may change the physical size and mode of plotting using additional data

xyplt *x y*

where \| *x* \| physical size (inches) of axis in x direction (\| *x* \| < 100),
\| *y* \| physical size (inches) of axis in y direction,
x , *y*
- +ve if log scale required,
- -ve if linear scale required.

The default data are: `xyplt  -8,8`

(ii) The 2D purejoy and contour plot option (*i.e.* | *p* | equal to 4, 5 or 6)
In the two-dimensional purejoy [Kubert et al. 1968] and contour plot [Trimble 1978] options, the calculated reaction rate or flux function $f_G(x,y,z)$, for collapsed group G, is plotted for a plane by keeping one of the coordinates constant. The data requirement for purejoy takes one of the following 3 forms:

plot	4	*k*	*angle*	i_1	i_2	j_1	j_2	0
plot	5	*j*	*angle*	i_1	i_2	k_1	k_2	0
plot	6	*i*	*angle*	k_1	k_2	j_1	j_2	0

where	*i*, *j* or *k*	define the x, y or z plane to be plotted *e.g.* if | *p* | = 4, then the required xy plane is given by the k^{th} z mesh point,
	i_1, i_2	define the x range from mesh point i_1 to i_2,
	j_1, j_2	define the y range,
	k_1, k_2	define the z range,
	angle	is the viewing angle (in degrees) measured anticlockwise, from the X (or R) axis for | *p* | =4 or 5 and from the Z axis for | *p* | =6.

If a complete plane is required, rather than a segment, the **plot** data should terminate with the *angle*.

Example:

```
plot  5, 3, 120, 0
```

Presentation of the plots may be changed as follows using the **purejy** and **ncycle** options.

purejy $x_1, x_2, \ldots, x_8$

where x_1 = length in inches (-ve) of y-axis of contour plot (default -8),
if $x_1 = 0$ a contour plot is not produced,

x_2 = (default 2) forces interpolation of function at *ix* equally spaced grid points in the x-direction,

where $ix = nxm \times \min\left\{ x_2, \sqrt{8}\,\frac{mx}{nxm} \right\}$

here mx = maxx for plot 4 or 5,
mx = maxz for plot 6,
nxm number of x (or z for plot 6) mesh points,

if x_2=0 then a purejoy plot is not produced,

x_3 = (default 2) forces interpolation of function, as above, but at grid points in the y-direction,

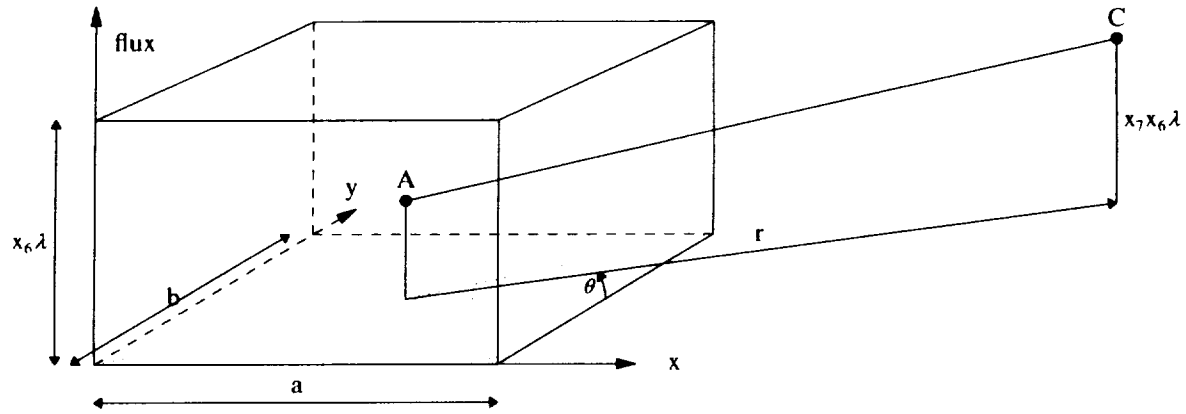
x_4 = horizontal size of frame for purejoy plot in inches (default 10),

x_5 = vertical size of frame for purejoy plot in inches (default 10),

x_6 = box height factor (default 0.7) sets the box height = $x_6 \times$ max (a,b),

x_7 = camera height factor (default 0.8) sets the camera height = x_7 $x_6 \times$ max (a,b),

x_8 = horizontal distance between camera lens and aiming point in half-diagonal length units (default 3) sets the camera distance, r = x_8 ($\frac{1}{2}\sqrt{\{a^2 + b^2\}}$).



Here a,b are the dimensions of the base of the box,
λ is max (a,b),
θ is the viewing angle,
C is the camera lens point,
A is the camera aiming point with the coordinates $(\frac{a}{2}, \frac{b}{2}, \frac{x_6\, x_7\, \lambda}{x_8 + 1})$.

For a contour plot the number of contours, *n*, per power of 10 may be set with

ncycle *n*

For ease of interpretation of contours, multiples of 5 are recommended for the value of *n* (default *n*=5).

An example follows of data required to produce contour and 2D purejoy plots of group 4 neutron flux.

```
groups 1 4,4
mreg 1(1)6
plot 4 1 120 0
edit -1 0 0
```

(iii) The 3D scientific visualisation plot option (*i.e.* | *p* | equal to 7)
Recently the advent of scientific visualisation packages, such as AVS [1994], has opened the way for 3D data to be inspected in new and interactive ways. The present visualisation option either creates data for such packages on disk or communicates directly. The feature is similar to **output vis** (Section 6.23) except that the range of output is extended to involve reaction rates including various weighting options. Further, like all edit features, it is only available on exit from a calculation, whereas **output vis** is available during a calculation by setting **iout**.

6.27 Channel Power Edits

The **edits** option uses subroutine **sub10** to calculate flux, power, reaction rates and weighted reaction rates averaged over channels or bulk regions of the reactor. Data used with this option, which include **lx,ly,lz** supermesh data for slicing the reactor, are listed below:

groups...
mxs...
lx *mx* $i_1, i_2, \ldots, i_{mx+1}$
ly *my* $j_1, j_2, \ldots, j_{my+1}$
lz *mz* $k_1, k_2, \ldots, k_{mz+1}$
edits *l m n p*

where *mx, my, mz* are the number of channels in the x (or r), y(or z), z directions respectively, and the lines i for $i_m \le i \le i_{m+1}$ form the m^{th} channel in the x direction, with similar meanings applied in the other directions for j and k,

and the **edits** data are the same as the point **edit** data (Section 6.26.2).

The following provides an example of data for printing flux, averaged over mesh intervals rather than the standard POW3D edge flux.

```
groups 1 1,4
lx   22  1(1)23
ly   16  1(1)17
lz   16  1(1)17
edits   -1,0,0,0
```

6.28 Perturbation Analysis

Perturbation analysis provides a formula for calculating the change in effective multiplication constant δk, resulting from slight changes (perturbations) to cross-sections of materials constituting the reactor. For changes which are not so small, δk may be calculated simply by carrying out two normal calculations (`calc=real, eigenv`), one with standard data and the second with perturbed data. This approach is adequate when δk is much greater than the accuracy requested (**acclam**, Section 6.22.1) and only a few cases are to be studied. For small perturbations however (δk of the order of say 10^{-5}) or when many cases are to be studied the perturbation analysis approach is necessary.

Central requirements for perturbation analysis are the real and adjoint fluxes of the unperturbed system. Using first order perturbation theory, the following expression can be derived for the change in effective multiplication constant δk consequent to a perturbation of cross-sections.

$$\frac{\delta k}{k} = \frac{\delta(\nu F)RR}{(\nu F)RR} - \left[\frac{\delta(REM - S)\,RR + \delta(LEAK)\,RR}{(\nu F)\,RR}\right](k + \delta k)$$

The reaction rates, RR, are defined as follow:

(i) fission emission

$$(\nu F)\,RR = \sum_g \chi_g \sum_{g'} \int_A \nu\sigma_{f,g'}\,\phi_{g'}\,\phi_g^*\,dV$$

(ii) the change in fission emission

$$\delta(\nu F)\,RR = \sum_g \chi_g \sum_{g'} \int_P \delta\nu\sigma_{f,g'}\,\phi_{g'}\,\phi_g^*\,dV$$

(iii) the change in removal minus scatter

$$\delta(\text{REM} - \text{S})\,\text{RR} = \sum_g \int_P \delta\sigma_{ag}\,\phi_g\,\phi_g^*\,dV - \sum_g \int_P \sum_{g'} \delta\sigma_{s,g\to g'}\,\phi_g\,\phi_g^*\,dV$$

$$+ \sum_g \sum_{g'} \int_p \delta\sigma_{s,g'\to g}\,\phi_{g'}\,\phi_g^*\,dV$$

(iv) the change in leakage

$$\delta\,(\text{LEAK})\,\text{RR} = \sum_g \int_P \nabla\phi_g^* \cdot \delta D_{n,g}\,\nabla\,\phi_g\,dV$$

The summations are over the set of all energy groups. The spatial integrals are calculated using finite differences. The fission emission volume integral is over all material regions A, whereas the other integrals are over the perturbed region P only. The following approximation of Robinson [1986] is used for the changes in diffusion coefficients to ensure that the leakage perturbation changes linearly with the transport cross-section.

$$\delta D = \delta\sigma_{tr}\frac{dD}{d\sigma_{tr}}$$

If data exist for material **inv455b** (Section 6.6.1), then $10^{-8} \times \sigma_{rem,g}$ (*ie.* $1/v_g$ with v_g expressed in units of cm sec^{-1}) is used to calculate prompt neutron lifetime in seconds, l, of the unperturbed system as follows,

$$l = (\text{REM})\text{RR}\ \ k/(\nu\text{F})\text{RR}$$

where (REM)RR is the integral over all material regions of $1/v_g$.

When studying (x,y) geometry systems with axial (z) leakage, then by default, the leakage is added to absorption. The axial leakage can be attributed correctly to a gross leakage term by use of the following keyword and data.

lreac *n*

where *n* is the number of the reaction in which the user has stored axial (z) $D_gB_g^2$, which has also been added to the removals.

The perturbation calculation is initiated by either of the keywords **perturb** or **aedit** and associated data.

6.28.1 perturb , region perturbation

The data for a region perturbation are entered as follow:

perturb $m_1, m_2, \cdots, m_n,\ \ r_1, r_2, \cdots, r_n$ **pend**

where m_i is the number of the material in the region to be perturbed,
r_i is the replacement material for material region number m_i.

For example, to calculate the perturbation caused by replacing material region 2 with material 10 and material region 3 with material 11 the following data would be used:

```
perturb  2,3  10,11  pend
```

A more complete example illustrates the feature applied to a perturbation in the MOATA calculation in which the thermal absorption cross-section of fuel is decreased by 1%. Note the need for an adjoint calculation and two extra materials which require a modified **prelude** to that given in Section 6.2. Details of running the calculation are given in Appendix F.

```
prelude maxx=23,maxy=17,maxz=17,maxg=4,maxm=6,maxf=1 end
 ...      ##(data of Section 6.2)
calc=adjoint,eigenv
start
xsd inv455b m(5) ## data for 1.+8/vel - needed for lifetime calculation
4*0
5.43478-2 1.42045+1 1.69205+2 4.18410+2
20*0
defn fuelv m(6) m(1)
xsd modify m(6) ## fuel with 0.99 times thermal abs xs
4*0
3*0,0.99
20*0
perturb 1 6 pend ## replace fuel by fuelv
```

6.28.2 aedit , cell perturbation

This option calculates the perturbation about a point and is convenient for detector response calculations. Cell perturbation data are entered as follow:

aedit *i j k m a*

where *i j k* specify the mesh numbers in the x,y,z directions respectively of the cell to be perturbed,

m is the detector material to be used for cross-section data,

a specifies that $10^{24}a$ atoms of detector material are to be included in the cell.

The calculated change in reactivity δk/k, per $10^{24}a$ atoms of detector material, is printed along with component terms. Note that if the reactor segment being considered is ½ of the entire reactor, and a reflective boundary condition is used, then the calculated reactivity effect δk/k is for the entire reactor which includes $2\times10^{24}a$ atoms of detector material.

6.29 Termination Command

Termination commands for POW3D calculations follow:

end occurring somewhere in the data (other than at the termination of **prelude**, **list** or DATRAN information) indicates the end of data for the present calculation. A new **prelude** and data for a completely independent calculation may follow.

stop causes POW3D to exit (that is to return to the AUSYS control program if run under AUS); **stop** is effectively equivalent to an end of file on input unit 1 and data after **stop** are ignored.

7. KINETICS CALCULATIONS

7.1 Introduction

The kinetics equations solved by POW3D were introduced in Section 2. The kinetics study usually starts with a steady state calculation and data needed for this were detailed in Section 6. Some neutron reaction data required for a kinetics calculation were also introduced (Section 6.10), as well as data to select the kinetics option (Section 6.17).

Usually for kinetics, the initial conditions are the steady state conditions

$$\frac{\partial \phi_g}{\partial t} = 0, \quad \frac{\partial C_i}{\partial t} = 0, \; S_g = 0, \; t \leq 0 \text{ for all groups.}$$

The POW3D flux normalisation to a total fission source of unity is somewhat arbitrary and prior to starting the time dependent part of the calculation, the code usually applies the normalisation

$$\sum_g \int_{\text{reactor}} f \, \sigma_{f,g} \, \phi_g \, dV = P$$

where P is the given power at t=0,
f is the fission energy release for each material (**fer**, Section 6.10.2), and
$\sigma_{f,g}$ is the (macroscopic) fission cross-section.

An exception arises when studying a purely time dependent external source problem, possibly including fissile material, where the initial conditions are the shutdown ones

$$\phi_g = 0 \, , \; C_i = 0, \; S_g = 0, \; t \leq 0 \text{ for all groups.}$$

To achieve these initial conditions, the kinetics calculation does not need to be preceded by a steady state calculation. If the kinetics calculation is preceded by other calculations then the following data may be needed (Section 6.20):

flx=0

One kinetics option allows calculation of the time variation in flux following a prescribed pulsed (feedback-free) variation of some, usually physical, parameter from the steady state value. Two types of reactivity variation are permitted. The eigenvalue $\lambda(t)$ may be :

(1) interpreted the same as λ in a criticality **search** (Section 6.18) and used to carry out physical changes to vary material concentrations (**sub1**), or mesh spacing (**sub2**), or control absorber positions (**sub3**), or

(2) applied as a weight to the effective number of neutrons produced in fission, and $\lambda(t) \frac{\nu}{k}$ then represents the response of the reactor to an unspecified physical change.

An alternative kinetics option allows the user to specify an external source pulse, and to follow the resulting time variation in flux. Further, both reactivity and source pulse options are permitted concurrently in the same calculation, but this feature is probably of limited use.

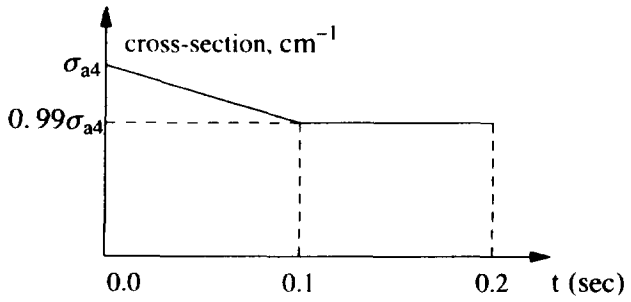
Notes:

(1) For 0D kinetics calculations, POW3D is slow and is not recommended since the overhead on data preparation is excessive.

(2) Future developments of POW3D could include feedback terms depending on the needs of POW3D users.

7.2 A Sample Kinetics Run (moata3d)

Consider the sample run of Section 6.2, extended to follow the flux variation resulting from a time pulse in the thermal absorption cross-section of fuel as sketched below. The kinetics calculation in the example below is preceded by the same steady state calculation as in Section 6.2, but additional prelude data are necessary. Details for running the calculation are given in Appendix F.




```
prelude ...,maxm=6,maxgd=1,maxf=1 end
...
start                                    - steady state data as in Section 6.2
* kinetics data follow
vel 1.84+9,7.04+6,5.91+5,2.39+5
igd=1                                    - only 1 delayed neutron group
betad=6.4-3
dlamda=0.08
sd                                       - delayed spectrum the same as prompt
defn fuel m(5)=m(1)                      - original fuel without change
defn fuelv m(6)=m(1)                     - set up fuel ready to modify cross-sections
xsd modify m(6)                          - change thermal absorption cross-section to 0.01σa4
4*1.-30                                    and remaining σ values to zero
3*1.-30,0.01
20*1.-30
search(-0.01)                            - concentration search
wsea 5*0,1
defn fuel wsea m(1)=m(5) 1 m(6) -1       - then σa4 (m(1))=σa4 (m(5))- 0.01 λσa4 (m(5))
                                           otherwise σf(m(1))=σf(m(5))
pulse(21)=linear (0,0),(0.1,1),(0.2,1)   - straight line segments describing reactivity pulse
reacp 21,1,0,0
dts 0,0.015,0.12
calc=real,kinetics
acclam=1.-3,accfo=1.-3
output data print
iout 0 1                                 - print output at every time step
start
* for asympotic reactivity calculation
calc=real,eigenv
acclam=1.-4,accfo=1.-4                   - restore usual accuracies
iout 0 0                                 - turn off print (otherwise every outer printed)
search(0)                                - turn off search
start                                    - steady state calculation with thermal group
stop                                       absorption cross-section in fuel of 0.99 σa4
```

Notes:

(1) In this example the neutron data for kinetics are given after the steady state calculation because delayed neutron fission spectrum data, which are different to the prompt, are not provided.

(2) Only 1 delayed group is considered here, although normally 6 would be used (with data available from Keepin [1965]).

(3) The required data pulse is set up using both the **xsd modify** feature (Section 6.11) and the post mixing feature available with the concentration **search** option (Section 6.18).

7.3 Overview of the Data Blocks for Kinetics

This section provides an overview of the data blocks required for a kinetics calculation and details follow in subsequent sections. Although the data blocks introduced previously (Section 6.3) for the steady state calculation are needed, only those specific to the kinetics calculation will be emphasised. The data blocks for kinetics consist of the following - essentially entered in the indicated order:

(6.4) **prelude** data - **maxf**, **maxgd** and **maxp**,
(6.9) Neutron data for kinetics - **vel**, **fer**, **igd**, **betad**, **dlamda** and **sd**(i),
(6.17) Calculation type - **calc** = **real**,**kinetics**,

(7.4) Initial power specification - not necessarily required,
(7.5) Time dependent pulse construction - of basic pulse shapes and pulse trains,
(7.6) Reactivity pulse insertion - as a train of pulses,
(7.7) Fixed external source pulse - as a train of pulses,
(7.8) Time integration step length - may be variable and must be user specified,
(7.9) Calculation termination conditions - accuracy (see also Section 6.22),
(7.10) Output requirement specification - to set content of output (see also Section 6.23),
(7.11) Calculation **start** - or **restart** from a flux dump (see also Section 6.24), and
(7.12) **edit** data - essentially as for steadystate (Sections 6.25 - 6.29).

7.4 Initial Power Specification

The initial power is specified with the data

power *p i*

where *p* $\begin{cases} = 0 \text{ then no change is made to the existing flux level, and} \\ \neq 0 \text{ is the power in watts (per cm for slab geometry) used for normalisation,} \end{cases}$

i $\begin{cases} = 0 \text{ then p}(\neq 0) \text{ is taken to be the required flux level,} \\ \neq 0 \text{ is the additional reaction number for } \sigma_f \text{ (3 for AUS files) which is used} \\ \quad \text{with the corresponding material fission energy release } \textbf{fer} \text{ (Section 6.9.2).} \end{cases}$

(Default values, `power 0.0 0`)

7.5 Time Dependent Pulses

Neutrons may be injected into a reactor system as pulses of reactivity, or as external sources, or both. The **reacp** option (Section 7.6) may be used for reactivity pulse insertion while the **scep** option (Section 7.7) may be used for application of a pulsed external neutron source. In this section the use of **pulse**(*n*) data to generate basic pulse shapes, to access standard pulse functions and to construct pulse trains is detailed. The pulse identification number, *n*, determines the pulse type as indicated below.

Pulse identification number, *n*	type of pulse
1	reserved for a user supplied **pulse** function
2-20	standard **pulse** functions available to the user
21-40	user generated **pulse** shapes and trains

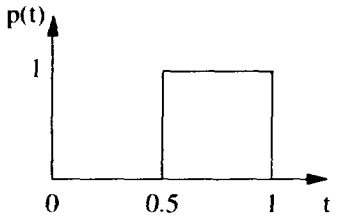
7.5.1 User generated pulse shapes

Options are provided for the user to generate basic pulse shapes which are either linear or stepwise. These pulse shapes, and the special functions in the next section, may be modified as required (Section 7.5.3). The definitions for basic shapes may be given in any order. Because the code interprets each definition as it is encountered, pulse shapes must be defined before they can be used to form a train. Each basic shape, or combination of shapes, is entered in the form

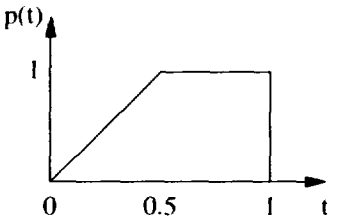
$$\textbf{pulse}(n) = \left\{ \begin{array}{l} \text{step} \\ \text{linear} \end{array} \right\} (t_1,p_1),(t_2,p_2),\ldots,(t_m,p_m)$$

where **step** produces a stepwise pulse,
linear produces a pulse by joining points with straight lines
n is the identification number of the generated **pulse** ($21 \le n \le 40$),
t_i time (in seconds) must be non-negative and entered in non-descending order,
p_i amplitude at time t_i.

Example:




```
pulse(21)=step (0,0),(0.5,1),(1,0)
```



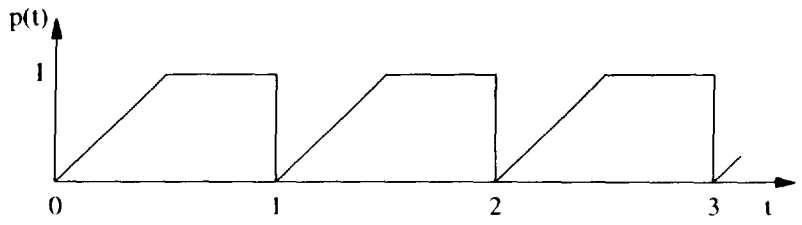

```
pulse(21)=linear (0,0),(0.5,1),(1,1),(1,0)
```

Here and elsewhere, all pulse types are treated as cyclic, so that the time within the cycle, t′, is given as:

$t' = t_m + (t - t_m) \text{mod}(t_m - t_1)$ for $t < t_1$,
$t' = t$ for $t_1 \le t \le t_m$,
$t' = t_1 + (t - t_1) \text{ mod } (t_m - t_1)$ for $t > t_m$.

The pulse is periodic and the period is normally $(t_m - t_1)$. The above example can be written more compactly and is illustrated below.

```
pulse(21)=linear (0,0),(0.5,1),(1,1)
```



7.5.2 Standard pulse functions

Some standard functions are also available as pulse shapes. They are identified by the **pulse** identification number $1 < n \leq 20$. These are continuous functions of time and are evaluated at discrete times to produce a table of values. The times must be defined, before the standard functions can be used, with the data

pulsetimes $t_1, t_2, \ldots, t_m$

where t_i are times in non-decreasing order.

The currently available standard shapes are listed below.

pulse(1) is reserved for a user supplied (real*4) **pulse** function in which amplitude is returned as p=**pulse**(t). The function must be written in the style of the user subroutines, Section 8.2, with 'common', *etc.* added by the POW3D preprocessor, or as a DATRAN function, Section 8.1, Datran(A.4)/(G).

pulse(2) = $\sin 2\pi t$

pulse(3) = $\cos 2\pi t$

pulse(4) = e^t

7.5.3 Pulse trains

Pulse shapes can be modified or combined using the following **pulse**(n) option,

pulse(n)= **scale** n', r [(t_f, p_f)] [(t_a , p_a)]

or

pulse(n) $\left\{ \begin{matrix} \textbf{prod} \\ \textbf{sum} \end{matrix} \right\}$ n′, n″

where one of **scale**, **prod** or **sum** must be provided and

n is the identification number ($21 \leq n \leq 40$) of the new train to be formed,

n', n'' are the identification numbers of defined pulses,

for **scale** r (>0) then the pulse shape n' is repeated r times,

t_f = scale factor for the time (default 1),

p_f = scale factor for the amplitude (default 1),

t_a = amount to be added to the time (default 0),

p_a = amount to be added to the amplitude (default 0),

for **prod** amplitudes of pulses n' and n'' are multiplied and

for **sum** amplitudes of pulses n' and n'' are added.

The **scale** option applied to the **linear** or **step** *type* **pulse**(n') function with the following data

pulse(n')= *type* (t_1,p_1),(t_2,p_2),.... (t_m,p_m)

pulse(n) = **scale** n' , r (t_f, p_f) , (t_a , p_a)

generates the new pulse train

pulse(n)=*type*(t_1, $p_f\, p_1{}' + p_a$) , (t_2, $p_f\, p_2{}' + p_a$) , . . .

where $p_1{}'$ is the appropriately interpolated pulse value from train **pulse**(n′) for $t' = t_f t + t_a$, *etc.*

For example the following statements,

```
pulse(21) = linear (0,0),(0.5,1),(1,1)
pulse(22) = scale 21,1 (0.7,0.3)
```

are equivalent to

```
pulse(22) = linear (0,0),(0.5,0.21),(1,0.3)
```

where, for example, at time `0.35` the linearly interpolated value for `pulse(21)` is `0.7` and the factor `0.3` gives a `pulse(22)` value of `0.21`.

The **prod** option applied to the two pulses n′ and n″, as in the data below

pulse(n') = *type* (t_1,p_1'), (t_2,p_2'),...,(t_m,p_m')

pulse(n'') = *type* (t_1,p_1''), (t_2, p_2''),...,(t_m, p_m'')

pulse(n)= **prod** n', n''

generates the new pulse train

pulse(n) = *type* (t_1, $p_1'p_1''$),(t_2, $p_2'p_2''$),...,(t_m, $p_m'p_m''$) .

If the times for the two pulse trains differ, the union of all available times is used. Values of p′ and p″ are obtained by appropriate (step or linear) interpolation if necessary. Extrapolation is unnecessary since the pulses are cyclic.

For example, a pulse train (n=23) consisting of 3 pulses of the type illustrated in (Section 7.5.1) with p(t) = 0, t > 3 can be generated with the following data in which `pulse(22)` acts as a mask for t > 3:

```
pulse(21)= linear (0,0),(0.5,1),(1,1)
pulse(22)= linear (0,1),(3,1),(3,0),(1.e20,0)
pulse(23)= prod 21 22
```

For example, since **pulse**(2)=sin $2\pi t$, the following data

```
pulsetimes = 0.(0.01)1.
pulse(21)  = scale 2,1 (0.5,0.1),(0.02,0)
```

would generate the pulse (evaluated at t = 0.,0.01,...,1),

pulse(21) = 0.1 sin 2 π(0. 5 t + 0.02).

As a further example to generate the pulse shape,

$$p(t) = 0.1\, e^{-0.01t} \cos 2\pi t,\ 0 \le t \le 5\ ,\ \text{zero otherwise,}$$

the following data are required.

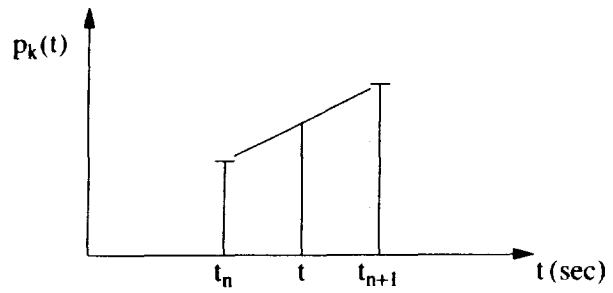
```
pulsetimes=0.(0.05)5.                  - t1 = 0, t2 = 0.05,...,t101 = 5
pulse(21)=scale 3,1 (1,0.1)            - p21(t) = 0.1 cos 2πt
pulse(22)=scale 4,1 (-0.01,1)          - p22(t) = e^-0.01t
pulse(23)=prod 21,22                   - p23(t) = 0.1e^-0.01t' cos 2πt' but with t'=t mod 5
                                         (that is the pulse repeats again for t > 5)
pulse(24)=step (0,1),(5,0),(1.e+20,0)  - mask for t > 5
pulse(21)=prod 23,24                   - as above but with p21 (t) = 0 for t > 5
```

The interpolation method, *step* or *linear*, is established from the given data thus:

```
(1) pulse(21) = step (0,0),(0.5,1),(1,0)   - as specified,
(2) pulse(22) = scale 21,2                 - interpolation method of pulse(21)
(3) pulse(23) = prod 21,22                 - interpolation method of pulse(21)
                sum ...                    - if both the same type or linear otherwise
(3) pulse(24) = prod 21,22 step            - as specified with trailing data
```

7.5.4 Advanced features

As a provision against minor roundoff errors when dealing with floating point arithmetic, interpolation includes 'extended' end points as in the exaggerated illustration for linear interpolation:



This is achieved using the variable $terpol_k$ (default 10^{-4}). Then if

$$(t_n - t)/(t_{n+1} - t_n) \le \min(terpol_k, 1/terpol_k)$$

$$p(t) = p_k(t_{n+1})$$

or if

$$(t - t_n)/(t_{n+1} - t_n) \le terpol_k$$

$$p(t) = p_k(t_n)$$

The default values may be changed, for example with the data

```
terpol = 100*1.e-5
```

where the excessive number of repeats allows for possible future extension and a value of zero must not be given. The somewhat surprising expression $\min(terpol_k, 1/terpol_k)$ arises from the simple expedience used by the code to force step interpolation by setting $terpol_k = 1/terpol_k$. Then the 'extended' end point to the right of t_n goes almost all the way to t_{n+1}. The set of times used by the standard pulse functions may be redefined as often as required. Say a user supplied function, $p_1(t)$, is only available to be calculated for the set of times 0.(0.2)1. The product $p_1(t)e^{-0.1t}$ may be calculated for a more reasonable set of times 0.(0.05)1. thus

```
pulsetimes = 0.(0.2)1.
pulse(21)  = scale 1,1
pulsetimes = 0.(0.05)1.
pulse(22)  = scale 4,1 (-0.1,1)
pulse(23)  = prod 21,22
```

Linear interpolation is used to extend `pulse(21)` to the union of the set of times of `pulse(21)` and `pulse(22)`, namely 0.(0.05)1.

A pulse train specification may include a reference to the same, previously defined, train. For example

```
pulse(21)  = linear (0,0),(0.5,1),(1,1)
pulse(21)  = scale 21,1 (0.7,0.3)
```

or even

```
pulse(21)  = linear (0,0),(0.5,1),(1,1) scale 21,1 (0.7,0.3)
```

Any memory used by the previously defined train is made available for reuse.

All **pulse** data may be cleared for independent **pulse** data thus

```
pulseclear
```

which restores the **pulse** table to its initial state.

It should be noted that the **pulse** data must all be contained in the **puls** array dimensioned with **maxp** (default value 50) in the user's **prelude**.

```
real*4 puls(2,maxp)
```

The array is equivalent to the array **ipuls**(2,maxp) and is used thus...

ipuls(1&2,1) = index for next **pulse** entry, length of index table (default 22)
ipuls(1&2,2) = index for **pulsetimes** entry, length of compacted **pulsetimes** definition
ipuls(1&2,3) = index for **pulse**(21), length of **pulse**(21) definition
ipuls(1&2,22) = index for **pulse**(40), length of **pulse**(40) definition
puls(1&2,23) = 1st **pulse** used t_1, 1st **pulse** used p_1
puls(1&2,**ipuls**(1,n-18)) = **pulse**(n,21 ≤ n ≤ 40) t_1, **pulse**(n,21 ≤ n ≤ 40)p_1
puls(1&2,**ipuls**(1,2)) = **pulsetimes** t_1, **pulsetimes** t_2 (or t_1 if t_2 is not given)

For the exponential weighted cosine example of Section 7.5.3, we must thus have at least

maxp=*sum of lengths(index+pulsetimes+pulse(21)+pulse(22)+pulse(23)+pulse(24))*

maxp=380 (*i.e.* 22+51+102+101+101+3)

7.6 Reactivity Pulse Insertion

Reactivity is inserted (or removed) following a variation of $\lambda(t)$. The λ variation is specified in the same way as for a criticality **search** (Section 6.18), except that for **search**(0) (default option), the effective multiplication constant is adjusted. Indeed the same **search** data must be provided, although a search is not performed. The user may vary one of the following:

(i) the concentration of a material in the reactor (**search**(-0.01)),

(ii) the size of indicated regions (**search**(-0.02)),

(iii) the positions of banks of control absorbers (**search**(-0.03)) as a function of time following a selected **pulse** train (Section 7.5.1) ,

(iv) the effective multiplication constant ($\lambda(t)$/k multiplies the fission emission),

(v) quantities coded by the user in **sub4**, ..., **sub9**.

Selection of the required reactivity option is made with the data

reacp $n\ r_2\ r_3\ r_4$

where n $\begin{cases} \neq 0 \text{ designates a } \textbf{pulse} \text{ train, } \lambda(t) = r_1 p_n(t) + r_3 + r_4 \lambda_s , \\ = 0 \text{ implies that forced reactivity variation is not required,} \end{cases}$

if r_2 $\begin{cases} \geq 0 \text{ gives } r_1 = r_2, \\ \text{otherwise } r_1 = \beta \lvert r_2 \rvert \text{ where } \beta \text{ is the total delayed neutron fraction,} \end{cases}$

and λ_s = eigenvalue from previous steady state calculation (for control absorber **search**).

(Default `reacp 0,1.,0.,0.`)

Note that when a criticality **search** is carried out, the eigenvalue multiplying say the concentration (**search**(-0.01)) or the mesh widths (**search**(-0.02)) is absorbed into data held in memory. (That is before a criticality **search** λ=1 and if after a **search** λ=2, then the concentration or mesh widths would be twice the values supplied by the user and λ would start again at 1 for any subsequent calculation). With the adjustment of control absorbers, however, (**search** (-0.03) - Section 6.18), λ specifies an absolute value rather than a relative value. POW3D does not correctly move fuel regions as the precursors do not move with the fuel.

The specification of $\lambda(t)$, the reactivity pulse, becomes clear. For a pulse about the critical position we require

$\lambda(t) = r_1 p_n(t) + 1$ (**search**(-0.01) or **search**(-0.02)),
or $\lambda(t) = r_1 p_n(t) + \lambda_s$ (**search**(-0.03)) (say control absorber moved out 10 cm from critical position).

For a pulse of a desired physical magnitude (say control absorber completely out of the reactor) we require

$$\lambda(t) = r_1 p_n(t) \ .$$

7.7 Fixed External Source Pulse

The spatial distribution and group dependence of the fixed external source is entered into the array **fsce** using either direct entry (**fsce**) or element entry (**fscel**) as for a **source** calculation - Section 6.19. Selection of the required **pulse** train to be associated with the source data held in the **fsce** array at the time of encountering the present keyword is made with the data

scep *n*

where *n* { ≠0 designates a **pulse** train,
=0 (default) implies that an external source is not required.

In addition the data held in the **fsce** array is written onto disk for ease of recall when required. During a kinetics calculation the array **fsce** contains all sources in addition to any external source. The time dependent external source is then given by

$$s(i, j, k, g, t) = fsce(i, j, k, g)\ p_n(t).$$

A simple example follows:

```
fsce 100000*0              - to set the array to zero if this is not the first kinetics calculation
fscel(1,1,1,1-2)=0.8,0.2   - to set source at centre to 0.8 in group 1 and 0.2 in group 2
scep 23                    - to associate external source with pulse train 23
```

7.8 Time Integration Step Length

The user must supply the time integration step lengths, δ t seconds, with the data

dts *n d T*

where *n* { ≠ 0 selects **pulse** (n), a train of all +ve amplitudes, then δ t = p_n(t) d,
=0, d≠0 sets δ t = d ,
=0, d=0 selects times from **pulsetimes** data,

T time at or beyond which the calculation is to terminate,

δ t ≤ 0 for any time encountered in the calculation (say at the end of **pulsetimes** data), then the calculation also terminates (as an error).

(Default **dts** 0 0 0)

For example the following time steps

$0 \le t < 0.01$, $\delta t = 5.\text{-}3$
$0.01 \le t < 0.1$, $\delta t = 0.05$
$0.1 \le t < \infty$, $\delta t = 0.1$

could be specified with the data

pulse(21) = step (0,5.-3),(0.01,0.05),(0.1,0.1),(1.e20,0.1)
dts 21,1,0.3

It is recommended that a step length δ t be chosen such that the power (or flux) does not increase by more than say 20% in the step. Larger time steps may be tolerable as the method used is stable.

7.9 Calculation Termination Conditions

The calculation termination conditions for kinetics are the same as for steady state calculations (Section 6.22). Since errors are introduced by the (generally large) time step, the default accuracies used for neutron balance (`acclam=1.-4`) and fission source (`accfo=1.-4`) are not usually warranted. After the steady state calculation the accuracies should perhaps be changed. The following are found to be usually adequate for kinetics calculations.

acclam=1.-3 **accfo**=1.-3

The user may also limit the number of outer (fission source) iterations per time step instead of the code default value of `nol=100`. If the limit is exceeded the code continues with the next time step. Usually changing **nol** is only necessary with zero-dimensional calculations where POW3D always uses 2 iterations to assess convergence. With zero-dimensional kinetics calculations the following data may save computer time:

nol=1

7.10 Output Requirement Specification

The output options available for kinetics calculations are the same as for steady state calculations (Section 6.23). With the kinetics calculation, however, output is not produced after each convergence of the flux (that is every time step) but only if the specific kinetics time limit or computer run time limit is exceeded. The **iout** option may be used to produce **output** at intermediate time steps. The following example would produce printed flux output for time steps 5,10,15,... and the final step. Flux output on disk would be produced for time steps 10,20,30,... and the final step.

```
output data print flux
iout     0     5     10
```

7.11 Calculation Start

A kinetics calculation is started, exactly as a steady state calculation, with **start** or **restart** (Section 6.24). Note that a flux dump is only produced on completion of a time step and hence a **restart** begins the step following the last completed time step.

7.12 Edit Data

Data may be edited (Sections 6.25 - 6.27) when the specified time limit is exceeded; that is, after the calculation is finished. To edit data at intermediate stages the calculation may be interrupted by specifying a time limit short of that ultimately required. After editing the calculation may be continued by specifying an increased time limit and by using the data

```
restart continue
```

which is equivalent to a **restart** from a disk flux dump except that information in memory is used.

Note: Data modified earlier needs to be restored by the user to the t=0 state before continuing if λ is built into, say, cross-section data.

8. ADVANCED FEATURES

Features are provided for coding by the user of extra options. A DATRAN compiler enables simple modification of intermediate data as it passes from one calculation to another within a POW3D run. More complex data modifications may require user supplied FORTRAN subroutines which are integrated with POW3D to create a temporary working version. Any user coding a FORTRAN subroutine would require an understanding of the way in which POW3D works and would need to ensure that the subroutine does not overwrite or otherwise interfere with POW3D. Some specialised DATRAN routines and data for specific applications are also provided and these are stored in a POW3D 'common' library. The user can access these from POW3D with the **call** statement.

8.1 Data Manipulation Compiler: DATRAN

The DATRAN compiler is restrictive but useful for minor processing of calculated data which is likely to involve only a few statements. A selected list of variables from POW3D common is used which better protects the integrity of a POW3D run than do FORTRAN routines. A brief overview of DATRAN follows:

- variables from POW3D common used in DATRAN must be selected with the **list** statement,
- an arithmetic or replacement statement is restricted to two variables or constants on the right of the '=' for example, `search(2)=akeff*0.97` , and may include functions,
- arrays may be indexed with a single variable or constant,
- functions are available and these may also be coded in the DATRAN language,
- control may be transfered to the next occurence of *keyword* in POW3D data with the statement, `return to` *keyword*,
- a record with an * in column 1 is treated as a comment as are all data on a record which follow ## ,
- any subroutine which can be written in FORTRAN (see next section) could also be supplied in DATRAN. This feature works through availability of default FORTRAN routines that can invoke DATRAN coding.

More detailed comments on DATRAN extracted from one of the POW3D routines may be accessed in the UNIX file `$AUS/Testcases/Datran`. On the *photon* computer at ANSTO, the environment variable $AUS should be set to `/home/gsr/aus`. In this report a reference such as Datran(A.4)/(H) refers to section (A.4)/H in the Datran file. An example follows with some comments on DATRAN included.

```
* DATRAN is a SKAN compiler/interpreter for a simple and much reduced
* form of FORTRAN. Its purpose is to enable the user to intermix data with
* coding and is designed to simply modify some existing data variables.
* An example would be to establish a POW3D search for an eigenvalue on
* the concentration of a component of the reactor system being
* studied, such that keff is to be 3% less than that of an immediately
* prior calculation.

prelude ... end
pow3d example1 - calculation of concentration eigenvalue
...
start ## 1st calculation
search(-0.01) ## normally k goes here but unknown when data prepared
* Now the example1 program written in DATRAN
  list akeff,search end ## list the required POW3D variables
  datran clg ex1 ## clg means compile, list and go - ex1 is progname
* example1 - calculation of concentration eigenvalue
  search(2)=akeff*0.97
  end

start ## 2nd calculation, etc.
```

The following examples are indicative of applications suitable for DATRAN coding.

(1) Modification of calculated data, stored in a specific array element, as illustrated in the previous example. The modified data can then be used in a subsequent calculation.

(2) Editing of printed output associated with k_{eff} and other critical eigenvalues. To print the percentage change of reactivity, from one calculation of a run to another, a few statements would suffice. For a more involved example, using a least squares fit approach, see Datran(A.3). The general DATRAN least squares coding can be accessed from the 'common' input library (Section 8.3.2) with `call poly`.

(3) A search parameter may be defined in terms of the critical eigenvalue. A routine may be incorporated into the **hifpow3d** version of POW3D (Section 6.17 and Robinson 1991a) which searches for the coarse control arm angle at which criticality is achieved in HIFAR. The only additional requirement is the following DATRAN subroutine **sub3** to replace the FORTRAN version.

```
      list wsea,aeigen end
      datran c1 sub3
      wsea(1)=aeigen
      return 1 ## return >0 to force recalculation of coefficients, etc.
      end
search(-0.03) fsea 0,20,15,56,2 ## 1st 4 are coarse control arm angles
```

(4) The set of available pulse functions for kinetics calculations may be extended using a DATRAN pulse routine. POW3D only permits linear, trigometric and exponential pulses for a kinetics calculation.

(5) Selected values of flux or other quantities may be printed at various stages of the calculation by invoking the printer dump routine, **pdump** - Datran(A.4)/(H). This routine was made available during code development and has been extended for regular use. The user may supply subroutines **pdump*n***, written in DATRAN, for $100<n<1000$ or use available default routines. The **idmp** parameter selects the stage at which to print the required values, as detailed in Appendix D4 of the POW report [Pollard 1974]. In the following example,

```
call pdump101
output pdump
idmp 7*0,101,12*0
```

the **call** loads the default DATRAN routine **pdump101** from the default 'common' library (Section 8.3.1). This routine prints the central flux for all groups after convergence of the flux solution (at stage 8 corresponding to the eighth element of **idmp**).

8.2 User Written FORTRAN Routines

A user may supply the following additional FORTRAN routines

sub4 sub5 sub6 sub7 sub8 sub9 uedit pulse

and use the temporary update feature (Section 4 and Appendix B2) to create a working version of POW3D. Other **sub*d*** subroutines, dedicated for specialised criticality searches (Section 6.18), which are not required for a search in the current job may also be replaced by the user. The routines should be stored in files with names of the form *routine*.u and be written essentially in FORTRAN but may include `cinsert` statements if data are required from the POW3D 'common' library (see next section) or to modify the calling sequence (Appendix B). The POW3D preprocessor, UPDATE2F, inserts COMMON in the routines and creates corresponding files named *routine*.f which contain the FORTRAN subroutines.

The **sub*d*** routines may be invoked for editing before or after a flux calculation with:

```
call sub9
```

The subroutine **uedit** may be invoked during, or at the termination of a calculation (Section 6.23), with:

```
output print,...,uedit
```

Some skeleton ideas are provided for writing these routines based on the `moata3d` example (Section 6.2) in which the thermal flux at the centre point is to be printed after convergence. The steps and data required to produce and run a temporary version of POW3D, with **sub9** included, are briefly indicated. The % prompt indicates that a UNIX command follows. The UNIX command, `more`, is included only to provide listings of input files, but otherwise is not required.

```
% more sub9.u

      subroutine sub9(kk)
cinsert common entry sident
c     prints centre flux for group 4
c     call readr(i,j,k,0)
c     i,j,k not 0 then since 3 (non-zero) args follow ndisk 3 of common data
c     call readr(flx,plane,group,0)
      call readr(  1,    1,    4,0)
cinsert call
c               flx(x,y,1)
      write(6,1)flx(1,1,1)
    1 format(/' centre flux, group 4 = ', 1pe12.3/)
      return
      end

% makemakepow3d
% make
% more moata3da

aus myjob << 'eof'
*dd1
step *
      testm pow3d=./pow3d
      link pow3d
      end
stop
*dd2
prelude maxx=23,maxy=17,maxz=17,maxg=4,maxm=4 end
pow3d moata3d xyz calc
...
start
call sub9
...
stop
eof

% moata3da >& moata3d.print & (or equivalent submission to a batch queue)
```

8.3 Routines and Data from the 'common' Library

The POW3D 'common' library is on FORTRAN unit 4, is maintained by the POW3D administrator, and can be read by the user. The 'common' library contains the following types of data.

(1) Data used by the POW3D preprocessor, UPDATE2F (Appendix B2), to set variable dimensions. These data consist of,

  (a) fixed common,
  (b) default prelude,
  (c) fixed common repeated,
  (d) variable dimensions,
  (e) variable-dimensioned common which provides an argument list.

(2) Data which are read by POW3D during execution and include default data for all variables. These data are followed by information from which the labelled `common/sident/` is generated as required for the preprocessor UPDATE2F (see Appendix B).

(3) An application specific data library provided for the user. This currently consists of only a few DATRAN routines and associated POW3D input data. The library may be expanded by the POW3D administrator as the need arises.

The function of the **call** statement has been extended to enable the user to read and use such type (3) data. The user need not be restricted to data supplied by the POW3D administrator on FORTRAN unit 4. The library unit and print option values may be changed from the default values `libcal 4,0`.

8.3.1 Call - to read 'common' library

The **call** statement, used to invoke routines written in FORTRAN, was introduced above. This statement is extended to enable a user to fetch data from a POW3D 'common' library. For example,

```
call hifpow3d (and no further trailing data for this statement)
```

establishes data for a HIFAR reactor version of POW3D (Section 6.17) and replaces the regular code version of **sub3** for control material concentration search (written in FORTRAN) with one that facilitates adjustment of the coarse control arm angle (written in DATRAN). The above statement is equivalent (Section 6.7) to:

```
read input(hifpow3d) lib on 4,0
```

8.3.2 Call - to execute DATRAN code

The call statement is extended to execute DATRAN code. For example,

```
call summary
```

executes the DATRAN code for **summary** (Section 6.23), if it has already been compiled and is equivalent to:

```
datran g summary
```

However if the compiled routine does not exist, an attempt is made to load it from the 'common' library on unit 4, compile it and then execute it.

9. CURRENT STATUS

The present, UNIX, version of POW3D was checked against the IBM/MVS version for about 40 different jobs, mainly of interest at Lucas Heights. Some of the runs were for kinetics calculations and some were calculated with POW as well. Agreement was achieved to most printed figures. (An original POW checkout was made by Pollard [1977].)

Future developments of the AUS scheme may include interactive (terminal oriented) input/output. To what extent POW3D will directly share in the interactive input/output is presently being considered. Also to what extent the code will be extended will very much depend on reactor studies of POW3D users.

10. ACKNOWLEDGEMENTS

The general encouragement and detailed help of fellow AUS module writers, particularly Mr Graham Robinson, is gratefully acknowledged.

11. REFERENCES

Advanced Visual Systems Inc. [1994] - AVS Manuals, Release 5. Waltham, MA, USA.

Argonne National Laboratory [1963] - Reactor Physics Constants, ANL5800.

Barry, J.M. [1982] - Multi-Dimensional Neutron Diffusion. Ph.D thesis, U. Wollongong.

Barry, J.M. and Pollard, J.P. [1977] - Method of Implicit Non-Stationary Iteration for Solving Neutron Diffusion Linear Equations. *Annals of Nuclear Energy*, **4**, p485.

Barry, J.M. and Pollard, J.P. [1979] - Applications of the Method of Implicit Non-Stationary Iteration (MINI) to 3D Neutron Diffusion Problems. *Annals of Nuclear Energy*, **6**, p121.

Barry, J.M. and Pollard, J.P. [1987 revised 1995] - AUS Diffusion Neutronics Module - POW3D, A Mathematical Description. AAEC/E612.

Bennett, N.W. and Pollard, J.P. [1967] - SCAN - a free input subroutine for the IBM360. AAEC/TM399.

Bondarenko, I.I. (ed.) [1964] - Group Constants for Nuclear Reactor Calculations. Consultants Bureau, N.Y.

Chiarella, C. [1969] - Calculation of resonance absorption in heterogeneous reactor systems. Ph.D. thesis, UNSW.

Doherty, G. [1969] - Some methods of calculating first flight collision probabilities in slab and cylindrical lattices. AAEC/TM489.

Francescon, S. [1963] - The Winfrith DSN programme. AEEW-R273.

Hansen, G.E. and Roach, W.H. [1961] - Six and sixteen group cross sections for fast and intermediate critical assemblies. LAMS-2543.

Hassitt, A. [1962] - A computer program to solve the multigroup diffusion equations. TRG-229(R).

Hopkins, D.R. and Oakes, D.B. [1968] - The two dimensional, multigroup diffusion code, GOG. AEEW-R532.

Keepin, G.R. [1965] - Physics of Nuclear Kinetics. Addison-Wesley, Reading, Mass.

Kubert, B., Szabo, J. and Giulieri, S. [1968] - The perspective representation of functions of two variables. JACM **15**(2), p193.

Meijerink, J.A. and van der Vorst, H.A. [1977] - An iterative solution method for linear systems of which the coefficient matrix is a symmetric M-matrix. *Math. Comput.*, **31**(137), p148.

Pollard, J.P. [1973] - Numerical methods used in neutronics calculations. Ph.D. thesis, UNSW.

Pollard, J.P. [1974] - AUS Module POW - A General Purpose 0,1 and 2D, Multigroup Neutron Diffusion Code Including Feedback-Free Kinetics. AAEC/E269.

Pollard, J.P. [1977] - AUS Diffusion Module POW Checkout - 1 and 2 - Dimensional Kinetics Calculations. AAEC/E387.

Pollard, J.P. [1978] - SKAN - A Free Input Labelled Output Variable Dimensioning Routine for the IBM360 Computer. AAEC/E431.

Pollard, J.P. and Robinson, G.S. [1969] - GYMEA - a nuclide depletion, space independent, multigroup neutron diffusion, data preparation code (IBM360 version). Unpublished, but an updated version of AAEC/E147.

Robinson, G.S. [1986] - EDITAR: a module for reaction rate editing and cross-section averaging within the AUS neutronics code system. AAEC/E621.

Robinson, G.S. [1987] - A Guide to the AUS Modular Neutronics Code System. AAEC/E645.

Robinson, G.S. [1991] - The Portable UNIX Version of the AUS Neutronics Code System. ANSTO unpublished document.

Robinson, G.S. [1991a] - 3D Diffusion Calculations of HIFAR Including the Coarse Control Arms and their Burnup. ANSTO/E703.

Stacey, W.M. Jr [1969] - Space - Time Nuclear Reactor Kinetics. Academic Press, N.Y.

Trimble, G. [1978] - XYPLOT - A Subroutine Package for the Computer Generation of Planar Graphs. AAEC/E437.

Wachspress, E.L. [1966] - Iterative Solution of Elliptic Systems. Prentice Hall, Englewood Cliffs, N.J.

Young, D.M. [1971] - Iterative Solution of Large Linear Systems. Academic Press, N.Y.

APPENDIX A - AUS MODULE ASPECTS OF POW3D

A1 System Aspects

The AUS modular neutronics scheme of computer codes [Robinson 1987 and 1991] was designed for general reactor calculations. The scheme consists of loosely linked reasonably self-contained modules which use well defined data pools (files) on disk. The modules may run as stand alone or as part of the AUS scheme.

The POW3D default associations (l_i, m_i, ddn_i) of FORTRAN units l_i to disk data files ddn_i and their aliases m_i (provided only for some ddn_i data files) are listed below.

(1,,dd2),	- POW3D input data which follows *dd2 record
(2,,dd12),	- 'card' image record output
(4,,dd53),	- common and standard POW3D data, `aus/pow3dcom`
(7,,dd15),	- alternative input used with `read input lib`
(8,lib),	- standard 200 group cross-section library, `aus/endfb6`
(9,,dd21),	- scratch
(10,xs1,dd33),	- cross-section data file
(11,xs2,dd34),	- cross-section data file
(12,,dd57),	- temporary 'card' image data file specified by the user as required
(14,,dd28),	- scratch
(18,xs3,dd40),	- cross-section data file
(19,,dd29),	- scratch
(20,,dd30),	- scratch
(21,,dd23),	- scratch
(22,,dd24),	- scratch
(23,,dd25),	- scratch
(24,fl1,dd36),	- POW-type flux dump
(25,,dd22),	- scratch
(26,gm1,dd35),	- geom data used with `read reg`
(61,fl3,dd41),	- POW3D-type (direct access) flux dump
(62,,dd17),	- (direct access) scratch
(63,,dd18),	- (direct access) scratch
(64,,dd19),	- (direct access) scratch
(65,,dd20),	- (direct access) scratch
(66,,dd26),	- (direct access) scratch
(67,,dd27)	- (direct access) scratch

The user may reset the default associations in the link statement of the AUS path program [Robinson 1987 and 1991]. The disk data files `dd2` to `dd10` are intended for user input data to a particular AUS module. In the example which follows, POW3D input is read from `dd9` instead of the default input data file `dd2`.

```
*dd1
step *
    link pow3d (1,9)        ##or  link pow3d (1,dd9)
    end
stop
*dd9
prelude...
```

A2 Data Pool Aspects

The data pools of the AUS system are simple structured (usually) sequential disk files which can be read using FORTRAN. The cross-section data file is specially compacted for easy skipping of materials and special AUS routines are provided to unbuffer the data. These special routines can also filter out data not interpreted by some modules. For a POW3D user the main interest here would be in the cross-section library and then probably only in the reactions for which cross-sections are available.

A material in an AUS cross-section file, in addition to scattering matrices ($P_0, P_1, ...$), can have the following (mostly cross-section) data with the corresponding AUS reaction numbers,

1	- σ_{tr},	transport cross-section,
2	- σ_a,	absorption cross-section,
3	- $\nu\sigma_f$,	fission emission cross-section,
4	- σ_p,	potential scattering cross-section,
5	- σ_{tot},	total cross-section,
6	- σ_f,	fission cross-section,
7	- σ_{b1},	first burnup reaction, usually capture (σ_γ),
8	- σ_{b2},	second burnup reaction or the (n,2n) reaction,
...		
9+NK	- D_r,	r or x direction anisotropic diffusion coefficient,
10+NK	- D_y,	z or y direction anisotropic diffusion coefficient, and
11+NK	- D_z,	z direction anisotropic diffusion coefficient for 3D.

Cross-section data for reaction 8 may be followed by kerma factors for NK (≤ 4) reactions and then by diffusion coefficients.

POW3D standard data consist of AUS reactions 1 to 3 (except that reaction 2 is the removal reaction), followed by the scattering matrix. These are followed by additional reactions and up to *maxs* (**prelude** data) of these are stored in memory by POW3D. The POW3D additional reactions, σ_{rn}, are read from the AUS file starting with AUS reaction 4 as listed below.

$\sigma_{r1} = \sigma_p$,
$\sigma_{r2} = \sigma_{tot}$,
$\sigma_{r3} = \sigma_f$,
$\sigma_{r4} = \sigma_{b1}$,
$\sigma_{r5} = \sigma_{b2}$.

For example, fission (σ_f) in the AUS cross-section data pool is reaction 6, but in POW3D it becomes σ_{r3}, which is additional reaction 3.

If diffusion coefficients, D_r and D_z, are available in the AUS cross-section data file then they are entered into the appropriate POW3D data array (dcx(i), dcy(i), Section 5.8.3) as well as a possible 3D set D_z (dcz(i)).

APPENDIX B - SKAN FREE INPUT ROUTINE AND POW3D UPDATE

B1 SKAN

All user input in POW3D is read by SKAN [Pollard 1978] which is a driver routine for the basic input routine SCAN [Bennett and Pollard 1967 and Clancy portable VSCAN unpublished notes]. SKAN is a keyword driven input routine so that data, for example,

```
sp  0.4 0.3 0.2 0.1 0. 0. 0. 0. 0. 0.
```

will be put into the **sp** (fission spectrum) array. The mode of the data may be mixed (it will be converted to that implied by the keyword) and special characters may be interspersed to aid checkout of the data. For example each of the following statements is equivalent to the above.

```
sp  4.e-1  3.-1  20.-2  .1  0.e0  0.e+0  0.d0  0.  0  0  ,
sp  0.4,0.3,0.2,0.1,6*0.
sp  0.4(-0.1)0.1,6*0
```

If insufficient data are supplied, that is another keyword or end of file interrupts the data, then the array is only partially filled (but would be initially zero).

B1.1 Width of record scanned

The data record is scanned from column 1 to column 72 and in addition all of the supplied input lists. It is possible for the user to change this by supplying the data

$opt *l,m,n* $

anywhere in the data beyond **prelude ... end**,

where l = first column of record to be scanned, usually 1,
m = last column of record to be scanned, usually 72($\leq$ 80),
n = listing option, 0 for no listing, $\neq$ 0 for listing.

B1.2 Data skipping

The following statement can be used to skip over the next two input items.

```
#2
```

For example

```
calc=real,eigenv,regcode
```

followed later by the data

```
calc=#2,hifpow3d
```

is equivalent to

```
calc=real,eigenv,hifpow3d
```

The skip feature needs to be used with care as some data (for example **xsd**) are initially stored in a temporary array. The use of the feature for such a case could produce unpredictable results.

B1.3 Access to variable data within POW3D

It is possible to access data from a variable in POW3D by preceding the keyword with the # symbol. In the following example

```
search(-0.02) #akeff,1.-3
```

the number stored in *akeff* will be used as the second numeric data of the **search** keyword. This feature could be used to carry out a 1D search calculation to give the same multiplication constant (*akeff*), as in a preceding 2 or 3D calculation. If, for example, akeff$\times$0.97 were required then DATRAN coding should be used (Section 8.1).

B1.4 Generating POW3D keywords

The **$mod** option, described in the SKAN report [Pollard, 1978], allows the user to generate new POW3D keywords and to enter data. The structure of the **$mod** data is illustrated in the POW3D 'common' which can be printed by including the word **printable** in the **prelude**. The **$mod** option may be used anywhere within POW3D data and is of the general form

$mod (*keyword, address, length, mode, option*),... $

where *keyword* provided by the user, should be different to variables in POW3D 'common',
address is the name of variable in which to store the data,
length of data in (*4) words,

mode
- 1 fixed point,
- 2 *4 floating point,
- 3 *8 floating point,
- 4 alphanumeric,

option
- 0 data are read only,
- > 0 options are available many of which require familiarity with POW3D source,
- 104 to 109 data are read and one of the user supplied routines sub4 to sub9 is called.

Several keywords may be defined between each **$mod** and **$**.

Consider a simple example in which data are supplied for a user written subroutine **sub9** (Section 8.2), followed by a call to the subroutine. The user could generate a keyword *sub9data* to store four (*8) items of data in the neighbouring temporary data variables **wa**, **wb**, **wc**, **wd**, of POW3D common. One way this could be achieved is by including the data which follow within a POW3D run.

```
$mod (sub9data,wa,8,3,0) $
 ...
sub9data 5,1.723d2,0.003,0
call sub9
```

Alternatively the following equivalent data could be used.

```
$mod (sub9data,wa,8,3,109) $
 ...
sub9data 5,1.723d2,0.003,0
```

B2 POW3D UPDATE

Variable dimensioning is used in POW3D because, in accomodating a range of different reactor models, preassigned dimensions for the many arrays would lead to considerable overestimation of the memory requirement. Even then, extreme cases could require recompilation of the entire source. Although FORTRAN 77 does not allow variable dimensioning in the main routine it does allow variable dimensions for arguments of subroutines with the implication that actual storage was assigned somewhere else. For the POW3D routines that need variable dimensions, the programmer would need to supply the extensive argument lists to pass the addresses of **flx**, *etc*. However this is very error prone and the resulting code is unwieldy to read. Instead a preprocessor, UPDATE2F, is provided that modifies simple argument lists by appending the full extensive list.

At runtime the variable dimensions are implemented by obtaining memory from the REGION= argument of the AUS run (Section 4) and assigning runtime dimensions for each array from **prelude** data provided by the user. The actual addresses of what appears as variable-dimensioned common are computed by SKAN and passed through the argument list.

Any remaining memory is assigned to the POW3D virtual input/output system, VIRTUL (Appendix C). The memory is fully utilised so that memory transfers take place as much as possible rather that input/output to disk. This can greatly speed up the code's performance, especially the elapsed time for execution.

In the subsection which follows the coding required by the POW3D preprocessor is briefly described. Procedures are described in Section B2.2 for producing a temporary version of POW3D. Procedures for updating the standard version of POW3D by the POW3D system administrator are described in Section B2.3.

B2.1 POW3D preprocessor UPDATE2F

The complete POW3D source is stored in files with **.u** as the extension. The UPDATE2F preprocessor creates intermediate FORTRAN files with **.f** as the extension. These are then compiled and object files are created with **.o** as the extension. Note that any changes to the code must be made to the **.u** files because the **.f** files are only intermediate files.

The routines in the **.u** files are coded in FORTRAN, usually with only a few arguments, and with pseudo comments, `cinsert...`, throughout the source. The pseudo comments should immediately trail the line of FORTRAN needing extended argument lists. These act as 'triggers' for the preprocessor to extend simple argument lists with data from the POW3D common library (Section 8.3).

The trigger `cinsert` may be followed by the keywords `common, entry, sident, call,`

where `common` adds the fixed common block to the subroutine coding (Section 8.3),
`entry` extends the argument list to include the variable-dimensioned common,
`sident` adds a labelled `common/sident/` which contains information needed by `cinsert call` and need not be specified in absence of `cinsert call`,
`call` extends the the variable-dimensioned common argument list and if used then the routine must have a matching `cinsert sident ....`

Note that `entry` and `sident` may not be required if only fixed common is required. However `common` must be supplied if `entry` and `sident` are used. Explanation of these features is also provided in the SKAN report [Pollard, 1978].

Examples are provided in Section 8.2 and in the following segments from the **uedit** routine.

```
      subroutine uedit(kk)   - note the single argument kk
cinsert common entry sident - sident is required in conjunction with cinsert call below
      ...                     to modify the subroutine, function or entry argument list.
      call kinet(0)
cinsert call                - extends the argument list in the subroutine kinet
      ...                     to include variable-dimensioned common
      end
```

To assist with understanding of the update routine, UPDATE2F, some brief help is provided. The routine is invoked as follows.

`update2f [show=on|off] [common=`*filename*`] `*this*`.u `*that*`.u ...`

where *this*.u *that*.u routines and common from *filename* are used to create FORTRAN routines *this*.f *that*.f
show=on|off for show of conversions during UPDATE2F (default `off`)
filename POW3D common library (default `./common`)

If an update is wanted but the files have .f extensions then files with .u extensions can be created with:

```
mvforu f u
```

If in doubt make a copy first. The above can be undone with:

```
mvforu u f
```

B2.2 Temporary POW3D update

The following three steps automate the POW3D update and compilation process to create a temporary working version of POW3D and should be invoked successively in the working directory.

(1) All user supplied routines should be stored in the working directory in files with **.u** extensions.

(2) **makemakepow3d** is a C shell script and should be invoked to produce a **Makefile** in the working directory.

(3) **make** is a standard UNIX command which should be invoked to update and compile POW3D. The preprocessor UPDATE2F is used. The FORTRAN **.f** files, the object **.o** files and the `pow3d` load module are all created in the working directory.

To use the temporary version of POW3D, the `link pow3d` statement in the AUS data must be preceeded with a `testm` statement as in the example below:

```
*dd1
step *
c                =/u28/jpp/.../pow3d  (a full path specification) may be used or
c                =./pow3d  which indicates pow3d in the working directory
      testm pow3d=./pow3d
      link pow3d
      end
stop
```

B2.3 Standard POW3D update - notes for the POW3D Administrator

The standard version of POW3D uses a library of FORTRAN compilation objects. The procedure to update the standard POW3D creates all updated object files in the library except for the main program which is created in the working directory. Examples of UNIX commands to create a **Makefile** for POW3D follow.

`makemakepow3d` - if default `prog=pow3d flags="-Ab -Sw -Em -Os"` are satisfactory,

`makemakepow3d prog=pow3dtemp flags="..."` - to make `pow3dtemp` in the working directory,

`makemakepow3d prog= flags="..."` - to update or create a library of POW3D objects,

`makemakepow3d prog= lib= flags="..."` - to make the main program object `main1.o`

Not all POW3D routines are successfully vectorised, or optimised, by the Fujitsu FORTRAN compiler. Hence POW3D has several directories for storage of source files, one for each set of different compiler flags, with a **Makefile** file in each separate directory.

APPENDIX C - FLUX DUMPS AND VIRTUAL INPUT/OUTPUT

C1 POW3D Flux Dumps

The code POW3D can read and write both POW3D and POW type flux dumps [Robinson 1987]. The disk data file which is used for the POW3D flux dump is also used for the storage of fragments of a flux dump (necessary for the virtual I/O discussed in the next section). Hence information for the management of user flux dumps needs to be specified.

The following example illustrates the data required, as part of an AUS run, to produce a POW3D flux dump. Usually the **info** default values are satisfactory and do not need to be provided.

```
aus myjob fl3=/dir/filename <<'eof'   -indicates file to be used for POW3D flux dump
prelude ... info  6,6,6 ... end       -data for management of flux dumps (see below)
...
output  fl3                           -forces a POW3D type flux dump
```

A good rule for a set of POW3D flux dumps is to use different files (with different **fl3** names) for different reactor calculations. Three dimensional flux dumps can use a lot of disk space and, unlike POW flux dumps, cannot be easily read to remove unwanted members. Management of flux dumps is done with the keyword **info** followed by 3 integers included with the **prelude** data.

prelude ... **info** *a,b,c* ... **end**

where *a*
- +ve is the number of the required **input** flux dump (which must match the problem),
- = 0 means the last dump will do,
- -ve then no input flux dump is used;

b is an **output** dump indicator,
if $b = a$ then flux dump '*a*' is overwritten,
if $b \neq a$ then a new flux dump is created at the end of a set of flux dumps;

c is an **output** dump indicator for a succession of dumps in the one run,
if $c = b$ then flux dumps created in the run all overwrite '*b*',
if $c \neq b$ then subsequent dumps accumulate consecutively at the end of the set of dumps.

The directory of dumps is established in such a way that dump number 1 is skipped. If for example,

```
info=-1,0,0
```

then no dump is read and any output dumps are written on top of each other as dump number 2 and any previous flux dumps in the file are destroyed.

The default data are

```
info=-2,2,2
```
and are taken to mean either,

```
info=-1,0,0
```
for a 'new' POW3D flux dump,

```
info=2,2,2
```
for an 'old' POW3D flux dump.

As an example, consider a collection of flux dumps represented as

fluxdumps 2,3,4,5

then

```
info 0,0,0
```

will read dump number 5 and write on it as well. However

```
info 3,0,0
```

will read dump number 3 and will create a new dump at the end, resulting in

fluxdumps 2,3,4,5,6

where dump number 6 will only be a partial dump unless

```
output fl3
```

is specified. Finally the user should note that the 'innocent' data

```
info 3,0,0
```

left in the data will add a new (complete or partial) flux dump every time the job is run. Therefore, before doing perturbation **edits** (Section 6.28), it is good practice to first create a copy of the good dump at the end with data for example

```
info 3,0,0
```

then for later runs to use the data

```
info 0,0,0
```

or, say,

```
info 6,6,6
```

which will always use dump number 6 even if later more dumps are added to the set.

C2 POW Flux Dumps

For compatibility with POW and other AUS modules, the code POW3D can read and write POW type flux dumps [Robinson 1987] unless the calculation is in three dimensions. The conventions for reading the dump follow those of POW. An example follows of data required, as part of an AUS run, to produce a POW flux dump.

```
aus myjob fl1=/dir/filename <<'eof'        - indicates a directory and file for the flux dump,
...
pow3d case1 ...
...
output  fl1                                - fl1 (or flux) forces a POW type flux dump.
```

The following is an example of data required to restart from a POW flux dump, provided the calculation is not in three dimensions.

```
prelude... info -1 0 0 ... end             - prevents reading a POW3D flux dump
...
restart case1
```

C3 Input/Output

All input and output in POW3D is carried out using the fast **drect** (direct access) routine [R. Cawley, unpublished] which is driven by the **readr** and **writer** routines. These routines can read and write either on disk or in memory if sufficient memory is available. Whether memory or disk is used does not affect the use of these routines. However POW3D is very slow if any of the I/O is to disk and it is therefore advisable to allocate sufficient memory for all I/O to be virtual. The remainder of this section is primarily of interest to a programmer intent on extending or modifying the POW3D program.

The code is equipped with standard and default data which are read before reading user supplied data. These include the dimension and other statements used by SKAN (Appendix B). Also included are data for the control of virtual input/output and these are listed in Table C1, with some keywords printed in bold to facilitate the ensuing explanation.

The POW3D **readr** and **writer** routines are called by,

or
call readr(*ithing,iplane,igroup,idisp*)
call writer(*ithing,iplane,igroup,idisp*)

where the arguments have the following meanings,

ithing
- -ve group to group transfers (see below),
- =0 see **readr** and **writer** options for multiple job dumps (Section C3.1),
- +ve is a pointer to an array in one of the segments **ndisk** 1, **ndisk** 2 or **ndisk** 3 and
  - if *iplane*=0 *i.e.* the 'thing' is not plane or group dependent
    igroup=0 then segment 1 of the table is used so that
    `call readr(20,0,0,0)`
    reads the **omega** array,
  - if *iplane*≠0 *i.e.* the 'thing' is not group dependent then
    igroup=0 segment 2 of the table is used so that
    `call readr(1,2,0,0)`
    reads the 2nd plane of the layout array, **mxy**,
  - if *iplane*≠0 then segment 3 of the table is used so that
    igroup≠0 `call readr(4,3,2,0)`
    reads the 3rd plane, 2nd group of the coefficient matrix, **cf1**;

iplane +ve is the required plane,

igroup +ve is the required group,

idisp indicates a storage position in common and if
- +ve then *idisp* is the number of array lengths from the storage address of *ithing* into which, or from which, data will be issued; thus
  `call readr(1,2,3,1)`
  will read **flx**, plane 2, group 3 into the array in common displaced by 1 from **flx**, which happens to be **wta** (normally *idisp*=0 is used),
- -ve then |*idisp*| is a pointer to segment 4 of the **ndisk** table, which points to the name of the actual array to be used for storage, thus
  `call readr(6,2,4,-3)`
  will read **fsce**, plane 2, group 4 into **wtb**, the 3rd entry following **ndisk** 4.

For a scattering matrix, with volume integrated group to group transfer data (*igfrom* → *igroup*), the first and last arguments of **readr** and **writer** must be negative. The storage block length is the same as for the data pointed to by *idisp*. The above routines are invoked through the following routines:

or
call reads(*-igfrom,iplane,igroup,-idisp*)
call writes(*-igfrom,iplane,igroup,-idisp*)

The FORTRAN input/output units for the variable arrays are assigned through the **ldisk** table. The **ndisk** table consists of four segments. Corresponding to each array in the first three segments of the **ndisk** table is an entry in the **ldisk** table, the absolute value of which added to 60 gives the FORTRAN file unit number (1 would refer to FORTRAN unit 60+1). If the entry is negative then any attempt to read data which has not been created returns all zeros. Thus the **flx** array is assigned to FORTRAN unit 61 which is the flux dump. The unit for the scattering matrix, with group to group transfer data, is the last entry of **ldisk**.

The I/O disk track pointers for the variable arrays are stored in the following tables:

idisk(*ithing*,1)	index for data which are not plane or group dependent,
jdisk(*ithing,iplane*,1)	index for plane dependent data ,
kdisk(*ithing,iplane,igroup*,1)	index for plane and group dependent data,
kdiskg(*igfrom,iplane,igroup*)	index for the scattering matrix.

TABLE C1

VIRTUAL I/O PORTION OF STANDARD DATA

```
*sizes for disk table check  idisk,jdisk,kdisk,next
idisze=50,20,20,20
*check for pickup from disk
icheck=#maxdsk,#maxz1,#maxggd,#maxx,#maxy,#maxg
*not plane or group dep dump on disk
ndisk 1 idisk,jdisk,kdisk,icheck,idisze,icheck,pow,calc,head,today
*         1     2     3     4       5      6    7    8    9    10
*          index tables   spare  szes-dsk&arr
        akeff,aeigen,bkeff,beigen,mop,dlam2,raccfi,raccfo,somega,omega
*         11    12     13    14    15   16    17     18     19     20
        no
*       21
*plane dep only
ndisk 2 mxy,scef,scefA1,scefA2,sce,wtb,riccg,diccg,siccg,yiccg,scefA3
*        1    2    3       4    5   6    7     8     9     10    11
*    layout  cheb extrap gpsce spare ...    4 iccg use    ...old4k
        scefA4,scefA5,scefA6
*         12     13     14
*globl   rbl     fs    scs
*plane and group dep
ndisk 3 flx,flxa,flx1,cf1,wta,fsce,del,delz,fsceA1,flxA1,flxaA1,cf1A1
*        1    2    3   4   5    6   7    8     9     10    11     12
*       flx adj  prec   spare exts mini     4kin old4scemini&modcf
*required for global rebalance (use with 'plane'=1)
  dgiccg,wtaA1,cf1A2,sceA1,flxA2,flxA3,cf1A3
*    13   14     15    16    17    18  19
*4iccg  plsce                wts old4rmini
*for core assignments to another destination (and group transfers)
ndisk 4 flx,wta,wtb,akeff,head,scef,fsce,sce,del,siccg,diccg,riccg
*        1   2   3    4     5    6    7   8   9    10    11    12
dgiccg,flxad
* 13     14
end
* -ve entries below read zeros when data not written
ldisk   21*1
         -2,-2,-2,-2,5,4,6,4,2,3,5
           -2,-2,-2
         -1,-1,-1,4,6,-2,2,3,4,5,5,6
           5,5,3,5,4,5,4
         7
*core priorities for virtul core
lcore   21*0
         6,7,6,6,7,6,7,7,7,7,6
           6,6,6
         8,8,7,7,6,7,7,7,6,6,6,6
           7,6,6,6,6,6,6
         0
*tries to put 1st segment in core (repeat means group as well as plane)
*others equivalent to 1st given unless core avail for them also
ncore 1  flx,sce,scef,mxy,del,delz,cf1,flx,del,delz,cf1
ncore 2  flx,flxa,flxa
ncore 3  flxa,flx1,flx1
ncore 4  sce,wtb,riccg,diccg,siccg,yiccg
```

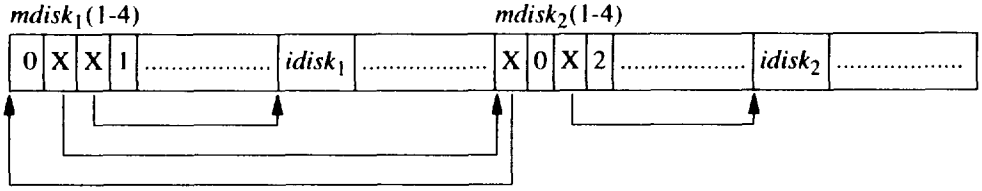
```
ncore 5   delz,dgiccg,dgiccg
ncore 6   scef,scefA1,scefA2,scefA3,scefA4,scefA5,scefA6
ncore 7   fsce,fsce
ncore 8   flx1,flxA1,flxA2,flxA3,flxA1,flxA2,flxA3
ncore 9   flxa,flxaA1,flxaA1
ncore 10  flx1,wta,wta,wtaA1,wtaA1
ncore 11  wta,sceA1,sceA1
ncore 12  fsce,fsceA1,fsceA1
ncore 13  cf1,cf1A1,cf1A2,cf1A3,cf1A1,cf1A2,cf1A3
end
```

C3.1 Reading and Writing Multiple Job Dumps

Each job dump has a master index **mdisk**, which is used to locate individual job dumps and includes disk pointers to the previous and next master indices. For a particular job, the pointers to data stored on disk are contained in three index tables **idisk, jdisk, kdisk** which are given relative to the master index.

mdisk(1) = disk pointer to last master index (or 0 if first),
mdisk(2) = disk pointer to next master index (or 0 if last),
mdisk(3) = disk pointer to *idisk*,
mdisk(4) = sequential dump identification number 1,2,...

Disk layout for FORTRAN unit 61 (the flux dump) is illustrated below.



Note that a 'null' disk consists of the 4 word record 0,0,0,0.

(1) **call readr(0,0,0,0)**

reads through the master chain until the last written (with *mdisk*(2)=0), whereas

```
call readr(0,0,0,1)
```

accepts the first master index. Consider the following instruction to read the third master index

```
call readr(0,0,0,3)
```

from the layout illustrated above; reading would stop at the last master index which is 2. The user may test for failure to read master index 3 with,

```
if(mdisk(4).ne.3)go to ...
```

The following statement will read the index **idisk**.

```
call readr(1,0,0,0)
```

(2) **call writer(0,0,0,0)**

writes a master index and sets the forward link in the preceding master index (now that the routine knows where the present master index is on disk). The dump identification number is set by the preceding **readr**, *e.g.*

```
call readr(0,0,0,1)
```

then

```
call writer(0,0,0,0)
```

writes in $mdisk_1$. To add a new dump to the end of the chain we would use,

```
mdisk(1)=0
call writer(0,0,0,0)
```

Note that for a new file, to create a master index in a locatable position on disk (and not a null record 0,0,0,0), the first master index must be **first** written on disk with

```
call writer(0,0,0,0)
```

even though the information in the master index is not yet up to date.

An example of the use of the routines in POW3D follows.

```
cinsert common entry sident
      ...
      call readr(0,0,0,0)
cinsert call
      if(mdisk(4).eq.0) go to ...   - bypass recovery because the record is null
c     file exists for recovery
c     read all index tables
      do 1 i=1,5
1     call readr(i,0,0,0)
cinsert call
      i1=ndisk(1)                   - length of first segment
      if(i1.lt.6)go to 3
      do 2 i=6,i1
    2 call readr(i,0,0,0)           - read akeff, etc.
cinsert call
    3 ...                           - here we would zeroise all pointers unless ldisk points to
      ...                             the recovery file 1, as other files assumed to be scratch
c     to create master index        - but initially with incorrect information
      call writer(0,0,0,0)
cinsert call
      ...
c     to finish
      j=5
      do 4 i=1,6
      call writer(j,0,0,0)          - writes out all index tables
cinsert call
    4 j=j-1                         - finishes with the master index updated
```

C3.2 Virtual Input/Output

The POW3D virtual input/output system was written before the availability of operating system virtual I/O. Because the POW3D virtual I/O system was tailor-made, it is much faster than than the operating system virtual I/O. POW3D attempts to put all I/O into memory but any data which do not fit in memory are transferred to disk. Priorities for data transfer to disk enable best use of memory to be made.

(1) The index tables are all dimensioned **idisk**(...,*maxdsk*). If `maxdsk=1` in the **prelude** then **readr** and **writer** are purely disk routines. However with the default value `maxdsk=2`, a virtual machine configuration is simulated (except that sharing of memory and swapping in and out of arrays are indicated by the user rather than on the basis of ad hoc rules). For virtual I/O, any memory beyond the last of the users common and up to the limit of the allocated memory is filled with memory buffers. The I/O pointers for the variable arrays are given by:

$$\textbf{idisk}(ithing,1) = \begin{cases} \text{disk track address, or} \\ \text{0 if not yet written,} \end{cases}$$

$$\textbf{idisk}(ithing,2) = \begin{cases} \text{*8 byte aligned memory address relative to 1st common, or} \\ \text{0 if no memory is available for this.} \end{cases}$$

Similarly for **jdisk** and **kdisk**.

(2) Memory buffers are assigned by the routine VIRTUL which should be called immediately after **prelud**. The VIRTUL routine uses **ncore** data (Table C1) firstly to assign memory and secondly to establish equivalences for arrays for which sufficient memory is not available so that memory can be shared by swapping arrays. The **ncore** data are entered in segments and processed in such a way that the actual equivalences used in a particular run depend on the segment in which memory ran out.

An attempt is made to assign storage blocks in memory for the arrays in the **ncore** table. Memory is allocated to arrays in the order they are listed until memory runs out. Some variable arrays are entered twice in an **ncore** segment (for example `flx` in the first segment, **ncore** 1). For the first entry of an array, memory is allocated by planes for the first group only. For the second entry of the array, memory is allocated by planes for all the remaining groups. When the first segment, **ncore** 1, is exhausted then VIRTUL assigns any remaining memory to arrays in the order listed in segments 2,3,... but, for this purpose, ignores the first array named in each of these segments.

If memory runs out for any array then an equivalence becomes effective between that array and the first array in the same **ncore** segment. In the second and subsequent **ncore** segments, the first array must (a) have appeared in a previous segment and (b) be for the largest storage block in the segment so that the other arrays, which are equivalent to the first, can share storage with the first (*eg* `flx`, in **ncore** 2). Note that `flxa` in segment **ncore** 2 is repeated to establish equivalence with the repeated `flx` in segment **ncore** 1 (the first by plane for one group and the second for all groups).

(3) The equivalence is managed by preceding the storage block with 4 indicators,

use ithing iplane igroup

where *use*
- +ve priority table pointer as in subsection (4),
- =0 means the block is empty,
- -ve the block is not up-to-date, and

ithing is a pointer to an array in one of the segments **ndisk** 1, **ndisk** 2 or **ndisk** 3 and,
iplane = 0 or 1 to specify the required index,
igroup = 0 or 1 to specify the required index.

The last three quantities are as in Section C3 for **readr** and **writer**.

(4) The priorities for storage of arrays in memory can be changed within a run by code authors but this feature has not been exploited. Priority of memory is indicated initially in the vector **lcore** where the correspondence with *ithing* is exactly the same as for **ldisk**. The priorities should lie in the interval 1 to 15 and normally would begin with

6 - for some unimportant quantities,
7 - for normal requirements,
8 - for some important quantities.

We may increment (or decrement) the priorities with the call,

call **prior**(*ithing,iplane,igroup,iprior*)

where the first three arguments are as in Section C3 for **readr** and **writer**,
ithing is a pointer to an array in one of the segments **ndisk** 1, **ndisk** 2 or **ndisk** 3, and
iplane = 0 or 1 to specify the required index,
igroup = 0 or 1 to specify the required index,
iprior = increment (usually 1 or 2).

The priorities of *things* sharing equivalent buffers can be dynamically changed and the calls can be made even if memory is only partially used.

(5) If the priority becomes -ve for example with

```
call prior(1,1,1,-100)
```

then any writes go to disk and memory is marked as not up-to-date by changing the sign of *use*. This feature is required by the 'flux dump' routine to force later recoverability.

APPENDIX D - FLUX SOLUTION OPTIONS

The report by Barry and Pollard [1987 revised 1995] details various POW3D options. Here a summary is given of the main options.

D1 3D Flux Solution

Three iterative methods are currently available for the 3D flux solution. These are:

(i) **mini** - Method of Implicit Non-stationary Iteration [Barry and Pollard 1977 and 1979],

(ii) **iccg** - Incomplete Choleski Conjugate Gradient [Meijerink and van der Vorst 1977].

(iii) **slor** - Successive Line Over Relaxation [Young 1971] and

A selection is made thus:

```
method=minimini

method=iccgiccg

method=slorslor
```

where the first word (4 characters) relate to the 2D (x,y) plane solution and the second word (4 characters) relate to the z between planes solution. Not all variations of the combinations of the words are possible. For example **iccg** may not be used as the second word unless also the first, otherwise combinations may be used.

Experience with the earlier IBM/MVS version of POW3D indicated that `minimini` was the best option particularly as it reduced the 'wall clock' time compared with other options. The default was therefore `minimini`. The VP2200/UNIX supercomputer version uses the same default.

The methods **mini** and **slor** use a solution method based on solving for a line at a time. The direction, say north-south, and sweep, say east to west, is established by POW3D to best suit the problem. However the user may provide a specific line-solution direction with data following the keyword **line**. The data and its interpretation follow.

```
line= 1  line s to n solved from w to e

line=-1  line s to n solved from e to w

line= 2  line w to e solved from s to n

line=-2  line w to e solved from n to s
```

Similarly the z-plane sweep is established by POW3D. Again the user may provide a specific sweep with the data **linez** which has the interpretation:

```
linez= 1  bottom to top

linez=-1  top to bottom
```

D2 Group Flux Solution

Two iterative methods are currently available for the group flux solution. These are:

(i) `methsc = mini`
(ii) `methsc = gs`

where **gs** refers to the usual Gauss-Seidel method. For extensive scattering **mini** accelerates convergence compared with **gs** and **mini** is therefore the default option.

APPENDIX E - RESONANCE CROSS-SECTION DATA FILE

Usually cross-section data for POW3D are read from an AUS file with resonance shielded data (Appendix A). The option to obtain neutron data from a resonance library is no longer used but is retained nevertheless. Hence resonance shielded cross-sections may be obtained from an AUS file such as the 26-group, 3-temperature resonance tabular ABBN library [Bondarenko 1964], or the 16-group, 1-temperature resonance tabular Hansen-Roach [1961] library.

Resonance absorption is calculated for a 2-region fuel-moderator model using an equivalence relation [Chiarella 1969]. The effective potential scattering cross-section for each material (including an allowance for resonance overlap) is calculated and the required cross section is obtained using both potential scattering ($\sqrt{\sigma_p}$) and temperature ($\sqrt{T}$) interpolation. User supplied input stream data may be used together with library data (with the same number of energy groups). User supplied potential scattering may be used with library data to influence resonance absorption.

Information can be extracted from a resonance library with the following data essentially in the order indicated:

xsd - the cross-section data selection,

homovr - the homogeneous volume ratios or homogeneous concentrations,

albar - the 2-region resonance equivalence relations information (chiefly $\bar{l}$ the mean chord length through the fuel region), and

read lib - selection of the required library and initiation of program action.

When reading two different resonance libraries, the influence of materials in the first library on resonance materials in the second library is correctly maintained through in-memory cross-sections, provided the same resonance material is not in both libraries.

Note: With a resonance tabular library it is necessary that in the **prelude** *maxs* ≥ 2.

E2 xsd, Cross-section data selection

Cross-sections for a particular material are selected from a resonance tabular library in the same way as from a standard AUS cross-section library (see Section 6.6). Some extra details follow:

xsd [[[*matname*] *source*] *mod*] **m**(*m*) [[[*conc*] *homovr*] *temp*]

where - cross-sections are multiplied by the homogeneous concentration, *conc* × *homovr*, - see also Section 6.6.2, and

- if several temperatures are entered for a material with **xsd** data then a homogeneous average temperature is used.

For example, neutron data may be selected from the old Hansen-Roach cross-section file as follows:

```
xsd  u238,hansen,orig   m(1) 0.04749, 0.1005
xsd  u235,hansen        m(2) 3.425-4, 0.1005
xsd  d                  m(3) 0.06655, 0.8995
xsd  o                  m(4) 0.03328, 0.8995
```

E3 homovr, Homogeneous volume ratios

Homogeneous volume ratios may also be included with user supplied input stream neutron data just as for a library material except that

[[[*conc*] *homovr*] *temp*]

are trailed by a second **m**(*m*) and cross-section data as in the example:

```
xsd   d   m(3)   1, 0.8995   m(3)
1.5028-1, 3.44185-1, 3.74146, 3.63482-1  ...
```

E4 albar, Equivalence relation information

POW3D uses a 2-region resonance equivalence relation for fuel and moderator [Chiarella 1969] which is permitted to degenerate to 1 region for normal homogeneous calculations. In the absence of user provided data, POW3D returns infinitely dilute cross-sections from libraries with resonance tabulations or otherwise previously shielded cross-sections (Appendix A).

The data requirement is:

albar $g,b,V\ \ \bar{l}_1\ ,\bar{l}_2\ ,\ldots\ \bar{l}_{maxm}$

where

g
- 0 slab,
- 1 cylinder (hexagonal pitch),
- 1.5 cylinder (square pitch),
- 2 sphere;

b
- 0 fully reflected system (*i.e.* a cell),
- ≠0 implies a reactor (cf. 0.71 used in geometry data, Section 6.15.1);

V
- ≠0 homogeneous volume ratio of fuel (resonance absorber) region to total
- 0 then homovr (k_1) is used instead as indicated below;

$\bar{l}_k$
- $\bar{l}$ (=4V/S) the mean chord length (cm) through the fuel region if m(k) is present in the fuel region,
- 0 if m(k) is not in the fuel region, *i.e.* in the moderator.

The default values for all **albar** data are 0 and imply a homogeneous system.

Notes:

1. The approximation in POW3D does not allow for more than 2 regions. Instead POW3D takes
   (a) $\bar{l} = \bar{l}_{k_1}$, 1st non-zero entry for which homovr(k_1)≠0 and
   (b) V = homovr(k_1) if V=0 is specified above.
2. A material (say ^{16}O), present in both fuel and moderator, is requested from the library twice as in the following example:

```
xsd o m(4) 0.03328,0.8995          - in moderator
xsd o m(5) 0.04102,0.1005          - in fuel
albar...0,2.53,...
```

As a simple example, consider natural uranium rods on a regular hexagonal pitch in D_2O. Assuming $\bar{l}$ = 2.53 cm, the data might consist of the following:

```
xsd    u238  m(1)  0.04749
xsd    u235  m(2)  3.425-4
xsd    d     m(3)  0.06655
xsd    o     m(4)  0.03328
homovr 2*0.1005, 2*0.8995
albar  1, 0, 0.1005   2*2.53, 2*0
read   lib on 8
```

APPENDIX F - A SAMPLE RUN (MOATA3D)

Data for sample runs were given for 3-dimensional steady state and kinetics calculations associated with the MOATA reactor (Sections 6.2 and 7.2 respectively). Here a summary of the input of a combined run is given. The manner of running the job is indicated and some brief output is presented. The data are stored in the file

$AUS/Testcases/pow3dreport

and the job is executed using the NQS batch system or run in the background thus

pow3dreport &

A DATRAN coded routine pdump111 (Section 8.1, Datran(A.4)/(H)) is used to reduce the normally extensive output from a run. It simply prints the flux, $flx(x_{ip}, y_{jp}, z_{kp}, g_{ig})$, and flux ratios compared to the previous time step, for the thermal flux at the centre of the fuel $flx(x_9, y_1, z_1, g_4)$ and at the top edge of the fuel $flx(x_9, y_6, z_7, g_4)$. The POW3D option **output print** is replaced by **output pdump** to achieve this.

The complete data for a run on the Fujitsu VP2200 supercomputer running UNIX follow. Sections of previously listed data are read using the **read input ... lib** option (Section 6.7).

```
REGION=30 aus $0 \
dd15=$0.lib \
dd12=$0.plot \
<< 'eof' 1> $0.print 2> $0.error
*dd1
step *
      link pow3d
      end
stop
*dd2
prelude maxx=23,maxy=17,maxz=17,maxg=4,maxm=6,maxgd=1,maxf=1 end
read input(pdump111)     lib on 7,0 ## datran code for printer dump
read input(steadystate) lib on 7,0 ## steady state data see Section 6.2
read input(pert)         lib on 7,0 ## perturbation data see Section 6.28.1
read input(kinetics)     lib on 7,0 ## kinetics     data see Section 7.2
stop
eof
```

Notes:

(1) The line of data that invokes the aus system is slightly more complex than given in the previous examples and wraps to several lines using '\'.

(2) Here REGION=30 sets the amount of memory requested by the job (30 MBytes). The shell variable $0 denotes the actual name of the file (*e.g.* pow3dreport) and consequently $0.print and $0.error are print and error files created by the run.

(3) The majority of data comes from input sections of the library indicated with dd15, pow3dreport.lib, provided in the working directory of the run. The perturbation calculation is unnecessary for the kinetics calculation to proceed and is given as a separate illustration.

(4) Data are saved on unit 2, corresponding to dd12, pow3dreport.plot, for later plotting of kinetics results using a PC spreadsheet program.

(5) POW3D uses column 1 print control for over-print, *etc.* and prints a line width of 132 characters.

A condensed listing from the output file is given. To assist recognition of what is being presented POW3D-type comments are included, but they are not part of the regular output. Perusal of the output indicates that the spatial flux shape, based on data for the two points printed, is not changing substantially with time. The case hardly warrants a 3-dimensional calculation but is used for convenience of presentation. It should be noted that the perturbation calculation estimation of k_{eff} for the asymptotic reactivity agrees extremely well with the result calculated directly.

```
--------------------------------------------------------------------------------
*** Steady state calculation corresponding to the initial state
 calc= real     eigenv   regcode
 outer   keff     sce-err  n-balance dominance crit-eigenvtn stage go-mins
   13  1.037188  0.0000725 1.000008  0.5588008 1.000000e+00  0   0     0.51 ...
       converged
 pdump111:flux (9,1,1,4) & flux (9,6,7,4)= 1.34794e-03 4.60000e-04

 calc= adjoint  eigenv   regcode
 outer   keff     sce-err  n-balance dominance crit-eigenvtn stage go-mins
   12  1.037193  0.0000257 1.000016  0.5291278 1.000000e+00  0   0     1.01 ...
 pdump111:flux (9,1,1,4) & flux (9,6,7,4)= 1.89601e-03 3.03577e-04

*** Perturbation calculation
 perturbation edit
 materials       1
 replaced by     6
                  keff  1.04566

*** Kinetics calculation
 calc= real     kinetics regcode
 outer   keff     sce-err  n-balance dominance crit-eigenvtn stage go-mins
    5  1.037193  0.0000709 1.000341  0.0000000 7.499993e-02  0   0     1.65 ...
       converged
    1 $$$$$$$$$$$      t,sec 1.50000e-02 p(flux) 5.90396e-04  period 4.27021e-01
 pdump111:flux (9,1,1,4) & flux (9,6,7,4)= 1.40184e-03 4.76337e-04
 pdump111:ratio(9,1,1,4) & ratio(9,6,7,4)= 1.03999e+00 1.03551e+00

   10  1.037193  0.0000635 1.000341  0.0000000 2.250066e-01  0   0     2.17 ...
       converged
    2 $$$$$$$$$$$      t,sec 3.00000e-02 p(flux) 6.49797e-04  period 1.56465e-01
 pdump111:flux (9,1,1,4) & flux (9,6,7,4)= 1.54866e-03 5.24194e-04
 pdump111:ratio(9,1,1,4) & ratio(9,6,7,4)= 1.10473e+00 1.10047e+00

   15  1.037193  0.0000769 1.000872  0.0000000 3.750131e-01  0   0     2.56 ...
       converged
    3 $$$$$$$$$$$      t,sec 4.50000e-02 p(flux) 7.45855e-04  period 1.08798e-01
 pdump111:flux (9,1,1,4) & flux (9,6,7,4)= 1.78202e-03 6.01751e-04
 pdump111:ratio(9,1,1,4) & ratio(9,6,7,4)= 1.15069e+00 1.14795e+00

   20  1.037193  0.0000523 1.000241  0.0000000 5.250189e-01  0   0     3.05 ...
       converged
    4 $$$$$$$$$$$      t,sec 6.00000e-02 p(flux) 8.87252e-04  period 8.64065e-02
 pdump111:flux (9,1,1,4) & flux (9,6,7,4)= 2.12619e-03 7.15826e-04
 pdump111:ratio(9,1,1,4) & ratio(9,6,7,4)= 1.19314e+00 1.18957e+00

   25  1.037193  0.0000658 1.000358  0.0000000 6.750235e-01  0   0     3.47 ...
       converged
    5 $$$$$$$$$$$      t,sec 7.49999e-02 p(flux) 1.09163e-03  period 7.23611e-02
 pdump111:flux (9,1,1,4) & flux (9,6,7,4)= 2.62224e-03 8.80839e-04
```

```
pdump111:ratio(9,1,1,4) & ratio(9,6,7,4)= 1.23330e+00 1.23052e+00

  30  1.037193  0.0000563 1.000701  0.0000000 8.250247e-01  0   0     3.96 ...
      converged
   6 $$$$$$$$$$$      t,sec 8.99999e-02 p(flux) 1.39384e-03  period 6.13760e-02
pdump111:flux (9,1,1,4) & flux (9,6,7,4)= 3.35852e-03 1.12466e-03
pdump111:ratio(9,1,1,4) & ratio(9,6,7,4)= 1.28078e+00 1.27680e+00

  35  1.037193  0.0001113 1.000634  0.0000000 9.666805e-01  0   0     4.33 ...
      converged
   7 $$$$$$$$$$$      t,sec 1.05000e-01 p(flux) 1.84502e-03  period 5.34901e-02
pdump111:flux (9,1,1,4) & flux (9,6,7,4)= 4.45609e-03 1.48885e-03
pdump111:ratio(9,1,1,4) & ratio(9,6,7,4)= 1.32680e+00 1.32382e+00

  41  1.037193  0.0000731 0.999535  0.0000000 1.000000e+00  0   0     4.73 ...
      converged
   8 $$$$$$$$$$$      t,sec 1.20000e-01 p(flux) 2.40666e-03  period 5.64438e-02
$$$$$$ kinetics time limit  ( 1.18500e-01 sec) exceeded $$$$$$$$$$$$$$$$$$$$$$$
pdump111:flux (9,1,1,4) & flux (9,6,7,4)= 5.80106e-03 1.94253e-03
pdump111:ratio(9,1,1,4) & ratio(9,6,7,4)= 1.30183e+00 1.30472e+00

*** Steady state calculation corresponding to the final state
 calc= real     eigenv   regcode
 outer   keff     sce-err  n-balance dominance crit-eigenvtn stage go-mins
    7  1.045674  0.0000447 0.999988  0.5908222 1.000000e+00  0   0     4.98 ...
       converged
 pdump111:flux (9,1,1,4) & flux (9,6,7,4)= 1.36166e-03 4.62227e-04
```

Index

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Y

Z