ANL/NSE-21/34

Ducted Assembly Steady-State Heat Transfer Software (DASSH)

User Guide

Nuclear Engineering Division

ANL/NSE-21/34

Ducted Assembly Steady-State Heat Transfer Software

Prepared by
Milos Atz, Micheal A. Smith, and Florent Heidet
Nuclear Engineering Division, Argonne National Laboratory

August 6, 2021

REVISION HISTORY

Name	Date	Reason for changes	Version
Milos Atz	2021-08-06	Initial preparation	0.0

SUMMARY

The Ducted Assembly Steady-State Heat transfer software (DASSH) is being developed at Argonne National Laboratory to perform a steady-state thermal fluids calculation to determine the coolant flow and temperature distribution for a hexagonal reactor core configuration with ducted assemblies. DASSH is intended to be used in the early stages of the reactor design process when assembly components can be undefined or are undergoing considerable design changes. DASSH provides a rapid assessment of the temperature and flow distribution to allow quick characterization of the system and identification of problem areas.

In DASSH, each assembly contains an array of wire-wrapped fuel pins that are responsible for the power production in the reactor. DASSH calculates the temperature distribution by balancing energy among adjacent, connected, coolant channels arranged around the fuel pin lattice. Power distributions are obtained from the Argonne Reactor Computational (ARC) suite or from user input. Users fix the inlet temperature and can choose to control the temperature change across the core or the flow rate in the assembly. This same type of problem is currently performed at Argonne National Laboratory using the legacy software SE2-ANL, which DASSH is intended to replace.

This document is a guide for DASSH users. It summarizes the core capabilities of DASSH and highlights differences between DASSH and SE2-ANL. Instructions for obtaining and installing DASSH are provided along with some guidelines and recommendations for newer Python users. It covers the structure of the input file, provides directions for running DASSH, describes the various output files produced by the code, and provides instructions for visualizing data. Examples demonstrating various DASSH options are included. DASSH is under active development, so it is anticipated that this guide will grow and evolve as the code is updated.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 Introduction

The Ducted Assembly Steady-State Heat transfer (DASSH) software is being developed at Argonne to determine coolant flow and temperature distributions for a hexagonal reactor core configuration with ducted assemblies [1]. DASSH is a full-core, steady-state subchannel code intended to be used in the early stages of the reactor design process when assembly components can be undefined or are undergoing considerable design changes. DASSH provides a relatively rapid assessment of the flow and temperature distributions to allow quick characterization of the system and identification of problem areas.

This document is intended to familiarize users with the DASSH code. Other documentation covers the theory and methodology [2] and the verification and validation testing (in preparation). The remainder of this section describes the type of problem solved by DASSH, summarizes the current capabilities of DASSH, and compares DASSH with SE2-ANL, the existing steady state thermal hydraulics code used at Argonne to perform the same functions for reactor design. Section 2 provides instructions on the installation procedure. In Section 3, the input file structure is described. Section 4 gives an overview of the code execution procedure and Section 5 describes the code output. In Section 6, examples are presented that demonstrate code input, output, and visualization.

1.1 *Description of the problem solved*

DASSH considers a reactor core comprised of ducted assemblies, each containing an array of wire-wrapped pins. The assembly coolant is divided into subchannels arranged around the pin bundle lattice. An example subchannel map for a 61-pin bundle is shown in Figure 1. In Figure 1, the coolant subchannels are numbered in red; duct elements are numbered in blue. DASSH has the capability to model double-ducted assemblies; in that case, the bypass gap coolant subchannel rings are indexed next, followed by those of the second duct.

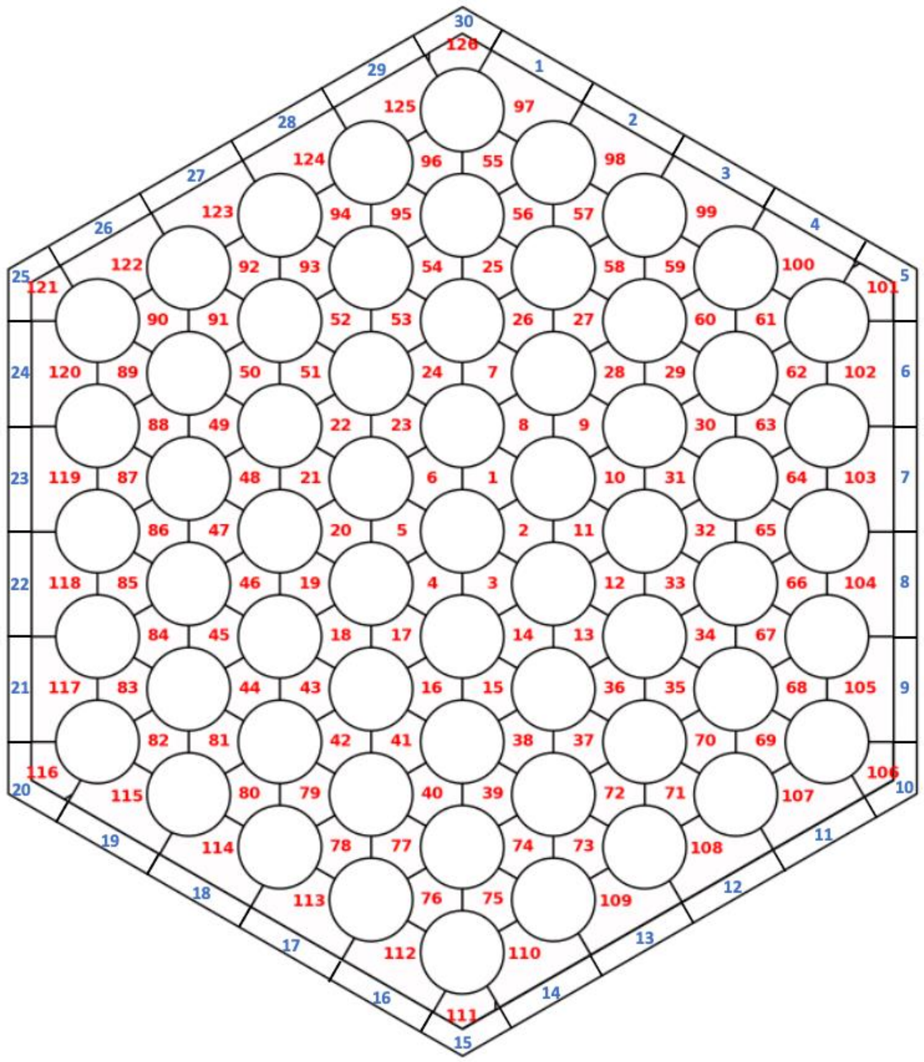


Figure 1. 37-pin bundle diagram and subchannel map

DASSH calculates the temperature distribution by balancing energy among adjacent, connected, coolant subchannels [3]. Thermal power is added to the pins, duct, and based on power distributions provided by the user or calculated from binary files produced by the DIF3D-VARIANT option of GAMSOR [4], part of the Argonne Reactor Computational (ARC) suite [5]. DASSH relies upon correlations to avoid solving the coupled equations for energy and momentum. In the DASSH model, it is assumed that heat is transferred axially by forced convection of coolant and radially (between subchannels) by a modified conduction term. This modified conduction term accounts for the turbulent mixing of coolant due to the wire wrap and is based on correlations related to pin bundle geometry and flow rate. Radial heat transfer across duct walls and in the inter-assembly gap occurs via conduction.

The inlet temperature is fixed and the user can select to control the temperature change over an assembly or the flow rate in the assembly. With that, along with the characterization of the assemblies and the power distribution, DASSH calculates coolant and duct temperatures by sweeping axially through the core. DASSH uses an axial forward-difference marching scheme, which simplifies the calculation but introduces an axial mesh size constraint to maintain numerical stability.

1.2 Overview of current capabilities and features

DASSH calculates full-core, steady-state coolant and duct temperatures and pressure drop across assemblies based on detailed power distributions. DASSH has been developed with many features to aid in the reactor design process, enumerated below.

- Users can define materials and specify temperature-dependent properties. Material properties can be updated with temperature throughout the calculation.

- Multiple correlations for friction factor, coolant flow split between subchannels, and mixing parameters between subchannels have been built into DASSH. Users can select different options for different assemblies, based on which is most accurate.

- DASSH calculates radial clad and fuel temperatures using a 1D infinite-cylinder shell model; the fuel can be subdivided into radial regions with variable characteristics that affect thermal conductivity.

- DASSH offers a low-fidelity model to reduce computation expense in regions that are not pin-bundles or in assembly positions that have low power and low mass flow rate. Pressure drop is calculated based on user-provided porous media parameters.

- DASSH includes built-in visualization capabilities of subchannel and pin temperatures for individual assemblies and for the entire core.

1.3 Comparison with SE2-ANL

DASSH uses a similar methodology to Argonne's existing steady state thermal hydraulics design software SE2-ANL [6]. SE2-ANL is a modified version of the SUPERENERGY-2 code [7] that obtains the pin power distribution via coupling to the ARC code suite. SE2-ANL also includes a pin-level temperature calculation and reactor hot spot analysis method where both nominal and two-sigma temperatures are calculated using the semi-statistical method [8]. DASSH is intended to be a rebuild of the SE2-ANL capability and thus preserve SE2-ANL capabilities while improving accuracy, usability, and sustainability. Like SE2-ANL, DASSH is applicable to reactors consisting of ducted assemblies containing wire-wrapped rod bundles. DASSH accounts for inter-assembly heat transfer, uses the same type of correlated parameters as SE2-ANL, and is coupled to the ARC code suite. DASSH has the following improvements over SE2-ANL:

- In DASSH, the convective heat transfer between duct walls and coolant can be modeled explicitly based on duct wall surface temperatures, whereas SE2-ANL employs a lumped thermal resistance between the coolant and the duct mid-wall temperature.

- DASSH features an improved model for double-ducted assemblies in which the two duct walls and the bypass gap between them are modeled explicitly. By contrast, in SE2-ANL, the inner duct was ignored and the flow rate to the pin bundle was reduced to 75% of the assigned value.

- DASSH updates material properties and correlated parameters at every axial step, whereas SE2-ANL uses material properties evaluated at a single average temperature for the whole core.

- The power distribution from neutron and gamma heating is obtained from DIF3D-VARIANT [4] whereas SE2-ANL uses DIF3D-FD [9] with a restriction on the triangular meshing. DASSH eliminates the SE2-ANL assumption that the radial pin power distribution in each assembly does not change axially. In DASSH, the axial power shape for each pin is obtained via the evaluation of the DIF3D-VARIANT basis functions. DASSH distributes neutron and gamma heating to the fuel, cladding, ducts, and coolant based upon the reaction rate distribution whereas SE2-ANL only allowed heating of the fuel and, optionally, gamma heating of the duct.

- With DASSH, users can assign different structural material properties in each assembly. Additionally, users may select from different built-in correlations beyond those available in SE2-ANL. DASSH can be extended to handle other coolant types (e.g. lead or molten salt) by including the coolant properties and adding different correlations as necessary, whereas this is not practical in SE2-ANL.

- DASSH eliminates the SE2-ANL assumption of a pin-lattice geometry throughout the modeled axial domain by allowing users to specify porous body medium input for low importance regions. This feature is particularly useful for early in the reactor design process where many parts of the reactor design might not yet be defined.

- To take advantage of the extensive data it generates, DASSH includes visualization capabilities on the pin, subchannel, and assembly levels.

- DASSH offers significant improvements in usability and maintainability over SE2-ANL. DASSH is written in Python 3 following software development best practices. The input and output are clearer than SE2-ANL and the code rigorously checks user input for mistakes.

2 Installation instructions

2.1 *DASSH pre-requisites*

DASSH is written for use with Python 3.5+. Users must a compatible version of Python. Installation instructions for Python are not covered here. It is recommended that users install Python through a Python package manager such as Conda (Anaconda [10] or Miniconda [11]). These tools centralize all things Python, making it easy to download and update dependencies as necessary. The Python utility "pip" is used to install DASSH. pip [12] is included in all Python versions later than 3.4. When installing DASSH, pip checks for all required Python packages (shown in Table 1) and will download and install them if necessary.

Table 1. Python packages required by DASSH

Package	Version	Purpose	Ref.
NumPy	1.18+	Required in DASSH primarily for its array operations	[13]
ConfigObj	---	Processes and validates DASSH input files	[14]
Pytest	---	Handles the DASSH unit and integration testing	[15]
Pandas	---	Required for reading correlation data in the DASSH tests	[16]
Matplotlib	3.2+	Required for visualization of DASSH results	[17]

2.2 *Obtaining DASSH*

DASSH is available open access on GitHub: https://github.com/dassh-dev/dassh. The code can be obtained by using the web interface to download the code or by using git commands, as shown below.

```
git clone <repo_site_address>
```

2.3 *Installing DASSH*

The installation copy of DASSH will have a directory structure of the following form:

```
src_DASSH/
├───dassh/
│   ├───correlations
│   ├───data
│   └───py4c
├───setup.py
└───tests/
```

Figure 2. DASSH directory tree

The “src_DASSH” directory is the root of the DASSH installation. The DASSH source code is located in the “dassh" subdirectory. Also within “dassh” are directories containing Python modules for calculating built-in correlations, built-in material property data, and a package (“py4c”) to couple DASSH with the ARC software suite. The “tests” directory contains the distributed test suite for DASSH. The “setup.py” file controls the installation.

To install DASSH, the following Linux command can be executed from the directory “src_DASSH” (the one that includes the “setup.py” file):

```
pip install .
```

2.4 Special cases and troubleshooting

This section provides recommendations for the DASSH installation and guidance on common issues that may arise during the process. All recommendations (e.g. using a virtual environment) are optional. For problem cases not described herein, new Python users are referred to the pip documentation [12] and may contact the developers with questions.

2.4.1 Using virtual environments

Package managers such as Anaconda [10] and Miniconda [11] also allow users to setup virtual environments, which enables the installation of DASSH and its dependencies on systems where users do not have administrator privileges, such as shared computing resources. Creating a virtual environment also allows users to insulate DASSH from changes in the global Python environment. Once installed, a virtual environment (in this case named “myenv”) can be created by invoking the following command:

```
conda create -n myenv python=3.8
```

Once created, the “myenv” environment can be activated and deactivated using the following commands:

```
conda activate myenv

# from within environment
conda deactivate
```

Options for specifying the Python version, packages to be installed, and more can be found in the documentation [18].

2.4.2 *Checking pip version*

The version of pip used to install DASSH must correspond to the Python 3.5+ installation that is used to run DASSH. Users can check which version of pip they are using by the commands shown below.

```
which pip

pip show pip
```

On computers with multiple versions of Python installed, Python 3 and the corresponding version of pip may possibly require unique commands for differentiation with default Python 2.x installations.

```
python3 -m pip install .
```

2.4.3 Installation in user-specified directory

Although using a Python package manager is recommend, some users may prefer not to do so. If not using a package manager, users may want or need to install DASSH in a directory of their choosing. This is relevant if installing on a remote computer without global administrator privileges. DASSH can be installed and called from the user's local directory using command line arguments to pip.

```
pip install . --target=<path/to/target/dir>
```

Installation in this way requires adding the target directory to your PYTHONPATH. To do so, use the following command:

```
PYTHONPATH=$PYTHONPATH:<path/to/target/dir>
export PYTHONPATH
```

This will need to be done every time you open a command line, so users are encouraged to add this to their .bashrc file.

2.5 Running tests

DASSH is distributed with a comprehensive test suite designed to ensure that each component of the code is performing as expected. Users should run these tests to ensure their installation was successful. To do so, from the root DASSH directory (which includes the "setup.py" file), run:

```
pytest tests
```

No tests should fail; it is okay for some tests to be skipped. If a failed test occurs, contact the developers to troubleshoot the installation as the failure may indicate a serious problem.

2.6 Updating DASSH

DASSH is still under active development and updates may be required to correct bugs and add capabilities. To update DASSH, users are recommended to replace or make a new copy of the distribution directory. The following Linux command, called from within the “src_DASSH” distribution directory to be installed as the new version, will update the installed version of DASSH.

```
pip install --upgrade .
```

3 Input file preparation

The DASSH input file is organized into blocks and subblocks, each of which is made up of multiple keyword-value pairs. Input files are read and validated using the ConfigObj library, after which DASSH applies additional consistency checks. In the ConfigObj format, input blocks are distinguished by square brackets and can be arranged in any order. Comments can be inserted anywhere in the file using the "#" character. The blocks of the input file are summarized in Table 2 and described in detail in the following subsections. The titles of named block sections (e.g. "Setup") are uppercase. In instances where users create their own sub-blocks, such as when defining assembly types, the names are formatted in lowercase in this user guide, but users may format these names as they like. Complete examples can be found in Section 6.

Table 2. Input block summary

Name	Description	Required
Setup	Controls DASSH environment, including I/O, various options to control the calculation, and information about dumping data to CSV.	No
Power	Inputs associated with DASSH power distribution, which may be defined by user or obtained based on binary files produced by DIF3D-GAMSOR.	Yes
Materials	User-defined materials and temperature-dependent properties.	No
Core	Specify core-wide parameters, including inlet temperature, power distribution options, and inter-assembly gap heat transfer model.	Yes
Assembly	Specify assembly pin-bundle geometric parameters and low-fidelity model region parameters.	Yes
Assignment	Assign assembly types and boundary conditions to assembly positions.	Yes
Plot	Request figure generation after DASSH execution	No

In the following subsections, tables are given to describe each keyword argument, including details for the type of value expected, whether it is required, options available for user-selection, and default values. In general, four different types of inputs are expected, depending on the argument. These are shown in Table 3.

Table 3. Expected DASSH input value types

Value type	Description

float	any type of real number (e.g. 130.2, 1.302E+2, and so on)
string	ASCII text, no quotation marks necessary
integer	any integer representation of a real number (e.g. 100, 1e2)
bool	True or False

Some inputs also allow or require lists of values, which may be input as comma-separated values. Where specified in parentheses, a single value is acceptable; otherwise, multiple values are required.

Finally, DASSH performs calculations in SI units; as such, all inputs should be provided in SI units. Table 4 codifies the expected units for each type of input parameter.

Table 4. Expected DASSH input units

Value	Units
Length	m
Temperature	K
Mass flow rate	kg/s
Power	W
Thermal conductivity	W/m-K
Density	kg/m^3
Heat capacity	J/kg-K
Viscosity	Pa-s (kg/m-s)

3.1 Setup block

The Setup input block allows the user to control problem environment options. The keyword arguments available in the Setup block and its nested subblock, Dump, are shown in Table 5 and elaborated upon in the remainder of this section.

Table 5. Input arguments to the Setup block

Keyword argument	Type	Req	Options and defaults	Description
axial_mesh_size	float	---	---	Minimum axial mesh size; will be ignored if numerical stability not satisfied
axial_plane	float (list)	---	---	List of floats indicating axial planes to include in sweep

log_progress	integer	---	---	Interval with which to report progress during axial sweep
conv_approx	bool	---	default=False	Enable modifications to duct wall convection model to relax mesh constraint
conv_approx_dz_cutoff	float	---	default=0.001 m	Axial step size below which DASSH automatically applies convection model modification.
calc_energy_balance	bool	---	default=False	Calculate intra- and inter-assembly energy balances
Subblock: Dump				
all	bool	---	default=False	All temperature results
coolant	bool	---	default=False	Interior and bypass coolant temperatures
duct	bool	---	default=False	Duct mid-wall temperatures
pins	bool	---	default=False	Radial pin temperatures
gap	bool	---	default=False	Inter-assembly gap coolant temperatures adjacent to each assembly
gap_fine	bool	---	default=False	Inter-assembly gap coolant temperature on gap mesh
average	bool	---	default=False	Assembly-average temperatures
maximum	bool	---	default=False	Assembly-maximum temperatures
interval	float	---	---	Spatial interval with which to dump data

One important feature available among the Setup keyword arguments is the capability to specify axial planes to be included in the sweep and to control the axial mesh size. The request is overridden if it is not satisfactory for numerical stability. The default minimum axial mesh size requirement is 0.01 m, which is set to ensure sufficient fidelity in the power distribution. The user can also choose to log the progress of the sweep with the “log_progress” option, which accepts as input a plane interval with which to report to the terminal. The progress log reports the number of planes that have been covered in the sweep and the elapsed time. An example of the progress log is shown in Figure 3.

```
DASSH....Progress: plane  650 of 1106; z = 2.27 m; cumulative sweep time = 00:02:58.22
DASSH....Progress: plane  700 of 1106; z = 2.44 m; cumulative sweep time = 00:03:11.92
DASSH....Progress: plane  750 of 1106; z = 2.62 m; cumulative sweep time = 00:03:25.63
```

Figure 3. Example of DASSH progress log

The keyword argument “conv_approx” enables an alternate model for heat transfer between the coolant and the duct wall. In DASSH, the default model calculates convective heat transfer with the adjacent coolant using the duct wall surface temperature. In assemblies with very low flow rates, this model can significantly impact the axial mesh size, in part due to the high value of the Nusselt number and heat transfer coefficient in liquid metals even at low flow rates. To ease this restriction, the “conv_approx” option enables DASSH to identify assemblies in which the coolant-duct heat transfer relationship limits axial stability and treat them with the SE2-ANL duct wall model, which uses convective and conductive resistances in series to connect the duct mid-wall with the adjacent coolant. The duct wall temperature calculation is unchanged; only the calculation of heat transfer from the duct to the coolant is modified. By including conduction through the duct wall, the model relaxes the axial mesh size constraint by up to an order of magnitude. The impact on the final result is minimal.

The Dump subblock allows the user to request detailed temperature data to be written to CSV files in the working directory. By default, no data is dumped. The user is encouraged to use the “interval” argument to specify the spatial interval at which data is dumped to the files. By default, data is written at every axial step, so this can relieve data I/O computational expense for problems with very small axial mesh sizes. Data is always dumped at the power distribution region boundaries obtained from the CCCC files and/or user input, and at region boundaries dividing the assembly in DASSH, as specified in the AxialRegion subblock of the Assembly input block.

An example of the Setup input block is shown in Figure 4. This input indicates a logging interval; enables the duct wall model modifications to relax axial mesh size constraints; and selects the types of data to write to CSV and the interval with which to write it.

```
[Setup]
    log_progress = 20         # Step interval to report progress
    conv_approx  = True
    [[Dump]]
        all       = False      # Dump all possible temperature data
        coolant   = True       # Interior and bypass coolant temperatures
        duct      = True       # Duct mid-wall temperatures
        pins      = True       # Radial temperatures for each pin
        gap       = False      # Gap temperatures adjacent to each assembly
        gap_fine = False       # Gap temperatures on the inter-assembly gap mesh
        average   = True       # Assembly-average temperatures
        maximum   = True       # Assembly-maximum temperatures
        interval = 10.0        # Dump data every 10 cm
```

Figure 4. Example Setup block input

3.2 Power block

The Power block is the means by which DASSH obtains core-wide power distributions from ARC binary files or from user input. Table 6 shows the arguments available for specifying the Power block input.

Table 6. Input arguments to the ARC block

Keyword argument	Type	Req	Options and defaults	Description
user_power	string	X*	---	Path to user-created CSV file with power distributions
total_power	float		default=None	Normalize total power (W)
power_scaling_factor	float		default=1.0	Scale total core power
Subblock: ARC				
power_model	string		• distribute (default) • pin_only	Indicate how to divide power among assembly components
fuel_material	string	X*	• metal • oxide • nitride	Fuel material category for power assignment

fuel_alloy	string	X*	• Zr • Al	If fuel_material=metal, indicate alloying element
coolant_heating	string	X*	• Na (or sodium) • NaK • Pb • Pb-Bi • LBE	Coolant material category for power assignment
pmatrx	string	X*	---	Path to PMATRX file
geodst	string	X*	---	Path to GEODST file
ndxsrf	string	X*	---	Path to NDXSRF file
znatdn	string	X*	---	Path to ZNATDN file
labels	string	X*	---	Path to LABELS file
nhflux	string	X*	---	Path to neutron NHFLX0 or NHFLUX file
ghflux	string	X*	---	Path to gamma NHFLX0 or NHFLUX file
* Requirement is conditional on other inputs				

The following subsections discuss how users can specify power distributions by either defining them independently or by referencing the binary data files produced by DIF3D. User-specified power distributions may be used alongside power distributions obtained from DIF3D. When both inputs are provided, the power distribution from DIF3D is processed first, then overwritten where user-defined power distributions are specified. This is useful for situations in which the DIF3D power distribution is inaccurate in a particular assembly or core region. In either case, the total core power can be normalized or scaled using the “total_power” or “power_scaling_factor” keyword arguments.

3.2.1 *Specifying DASSH power distributions*

DASSH accepts user-defined power distributions via a CSV file. The CSV file contains the power profile coefficients (for a polynomial in z) for each pin, duct, and coolant mesh in each assembly. After performing input checks and conducting any requested renormalization, DASSH uses these power distributions directly.

The structure of the CSV is shown in Table 7. The axial boundaries input to columns 3 and 4 are absolute axial positions in the assembly, between which the power distribution applies. The power distribution axial region definitions must be the same for all pins, duct elements, and coolant subchannels. No gaps or overlaps are allowed between regions. The coefficients input in columns 6+ are for polynomials on the [-0.5, 0.5] basis. Because this length basis is unitless, all should have units of W/m. The coefficients are zero by default; for example, for the user to specify a power distribution with no power distributed to the duct or coolant in a particular assembly, they may omit those rows from the CSV.

Table 7. CSV structure for user-specified power distributions in DASSH

Column	Value
1	Assembly ID
2	Material type (for pin, =1; if duct, =2; if coolant, =3)
3	Lower z-boundary (m)
4	Upper z-boundary (m)
5	Material ID (e.g. pin, duct element, or coolant subchannel index)
6	0th-order polynomial coefficient a_0 (for use as $a_0 z^0$)
7	1st-order polynomial coefficient a_1 (for use as $a_1 z^1$)
8	2nd-order polynomial coefficient a_2 (for use as $a_2 z^2$)
9+	Higher-order polynomial coefficients

An example of a user-defined power distribution is provided for a 7-pin assembly in Figure 5. The assembly is 3 m long and is split into three 1-m axial regions. The top and bottom regions produce no power, and the power distribution in the middle region is parabolic in z (centered at the region midpoint) and radially flat. To modify the relative pin powers, one could apply a scalar factor to the coefficients, thereby scaling the polynomial as desired. Because no inputs are provided for the duct elements or coolant subchannels, no heat is generated in the duct or coolant.

```
1, 1, 0.0, 1.0, 1,      0.0, 0.0,       0.0
1, 1, 0.0, 1.0, 2,      0.0, 0.0,       0.0
1, 1, 0.0, 1.0, 3,      0.0, 0.0,       0.0
1, 1, 0.0, 1.0, 4,      0.0, 0.0,       0.0
1, 1, 0.0, 1.0, 5,      0.0, 0.0,       0.0
1, 1, 0.0, 1.0, 6,      0.0, 0.0,       0.0
1, 1, 0.0, 1.0, 7,      0.0, 0.0,       0.0
1, 1, 1.0, 2.0, 1, 545833.3, 0.0, -550000.0
1, 1, 1.0, 2.0, 2, 545833.3, 0.0, -550000.0
1, 1, 1.0, 2.0, 3, 545833.3, 0.0, -550000.0
1, 1, 1.0, 2.0, 4, 545833.3, 0.0, -550000.0
1, 1, 1.0, 2.0, 5, 545833.3, 0.0, -550000.0
1, 1, 1.0, 2.0, 6, 545833.3, 0.0, -550000.0
1, 1, 1.0, 2.0, 7, 545833.3, 0.0, -550000.0
1, 1, 2.0, 3.0, 1,      0.0, 0.0,       0.0
1, 1, 2.0, 3.0, 2,      0.0, 0.0,       0.0
1, 1, 2.0, 3.0, 3,      0.0, 0.0,       0.0
1, 1, 2.0, 3.0, 4,      0.0, 0.0,       0.0
1, 1, 2.0, 3.0, 5,      0.0, 0.0,       0.0
1, 1, 2.0, 3.0, 6,      0.0, 0.0,       0.0
1, 1, 2.0, 3.0, 7,      0.0, 0.0,       0.0
```

Figure 5. Example demonstrating user-defined power distribution for 7-pin assembly

3.2.2 Obtaining power distributions from ARC binary files

To obtain power distributions from ARC binary files, the ARC subblock must be included. There, the user specifies the path to each CCCC binary file required by DASSH, listed in Table 6. Either the absolute file path or a path relative to the input file are acceptable. A utility program distributed with DASSH, called VARPOW, reads the binary files to divide power among materials designated as fuel, structure, and coolant. The user indicates material types for VARPOW via the remaining keyword arguments in the ARC subblock. The "fuel_material" and "fuel_alloy" inputs describe the fuel material for assignment of power. If "fuel_material" is set to "metal" (for example, U-Pu-Zr10 fuel for SFR) then the alloying element – either Zr or Al – must be specified as well. The "coolant_heating" input does the same for the coolant. If "coolant_heating" is not specified, DASSH will try to use the input given in the Core input block to the keyword "coolant_material".

An annotated example is shown in Figure 6 below, which shows that the binary files are in the same directory as the input file. Currently only full core problems are accepted, but support for periodic problems will be added in the future. It is important to note that only the scalar neutron and gamma flux binary files from DIF3D-VARIANT are required. Users can reduce the execution time of DASSH by providing the NHFLX0 file rather than the NHFLUX files. The GHFLX0 is simply the NHFLX0 file generated from the gamma flux calculation in GAMSOR [19].

```
[ARC]
    power_model     = distribute
    fuel_material   = metal
    fuel_alloy      = zr
    coolant_heating = sodium
    pmatrx          = ./PMATRX      # relative/path/to/PMATRX
    geodst          = ./GEODST      # relative/path/to/GEODST
    ndxsrf          = ./NDXSRF      # relative/path/to/NDXSRF
    znatdn          = ./ZNATDN      # relative/path/to/ZNATDN
    labels          = ./LABELS      # relative/path/to/LABELS
    nhflux          = ./NHFLX0      # relative/path/to/NHFLX0
    ghflux          = ./GHFLX0      # relative/path/to/GHFLX0
```

Figure 6. Example ARC block input

3.3 Materials block

Users can specify custom material properties with the Materials input block, arguments for which are shown below in Table 8.

Table 8. Input arguments to the Materials block

Keyword argument	Type	Req	Options and defaults	Description
Sub-block: [material name]				
thermal_conductivity	float (list)	X	---	Thermal conductivity (in W/m-K)
heat_capacity	float (list)	X*	---	Heat capacity (in J/kg-K); required only for coolant
density	float (list)	X*	---	Density (in kg/m^3); required only for coolant

viscosity	float (list)	X*	---	Viscosity (in Pa-s); required only for coolant
from_file	string	---	---	Relative path to CSV containing material properties
* Requirement is conditional on other inputs				

All properties (density, viscosity, heat capacity, and thermal conductivity) are required for coolant materials. The only required property for structural materials is thermal conductivity. Temperatures must be in units of Kelvin. Users may specify properties in one of two ways. First, users may input coefficients to polynomial correlations for material properties in terms of temperature (K), arranged in increasing polynomial order. If static values are desired, a single value can be assigned. An example is shown below.

```
[Materials]
    [[custom_coolant]] # format is a + bT + cT^2 + dT^3 + …
        thermal_conductivity =   109.75,    -0.065,  1.17e-5    # W/m-K
        heat_capacity        =  1437.17,     -0.58, 4.62e-04    # J/kg-K
        density              =    950.1,     -0.23, -1.46e-5,  5.64e-9  # kg/m3
        viscosity            = 8.64e-03, -5.01e-05, 1.14e-07, -1.2e-10  # Pa-s
    [[custom_structure]]
        from_file = /material/properties/file.csv
```

Figure 7. Example of custom Materials input block

Alternatively, users may provide material property data in a CSV table, pointed to by a relative path using the “from_file” input. The formatting is fixed: the first column must contain temperatures (K) in increasing order; the columns must have headers named as the expected property. A limited example for water properties is shown below; note the underscores in the header names for “heat_capacity” and “thermal_conductivity”. Missing values are allowed, as shown by the empty commas, or they can be filled with zeros. DASSH linearly interpolates these tabulated values to obtain continuous properties as a function of temperature; the missing/zero values are ignored in the interpolation for that property. The value of properties at temperatures above or below the given temperature range is set equal to that at the highest or lowest temperature specified, respectively.

```
temperature,density,viscosity,heat_capacity,thermal_conductivity
273.15,999.85,0.00179,4.2199,0.00422
277.15,999.97,,,
283.15,999.70,0.00131,4.1955,0.00419
288.15,999.10,,,
293.15,998.21,0.00100,4.1844,0.004157
298.15,997.05,0.00089,4.1816,0.0041379
```

Figure 8. Layout of CSV file containing material properties

3.4 Core block

The Core input block describes core-wide characteristics for the reactor to be modeled in DASSH. A description of each of the input arguments is provided in Table 9.

Table 9. Input arguments to the Core block

Keyword argument	Type	Req.	Options and defaults	Description
assembly_pitch	float	X	---	Assembly hex pitch across flats
length	float	X	---	Total core axial length
coolant_material	string	X	---	Coolant material properties
coolant_inlet_temp	float	X	---	Coolant inlet temperature
gap_model	string	X	• flow • no_flow (default) • duct_average • none	Inter-assembly gap heat transfer model
bypass_fraction	float	X*	default=0.0	If gap_model=flow, input fraction of coolant flow assigned to gap
* Requirement is conditional on other inputs				

The "coolant_material" input assigns material properties to the coolant. The coolant requires correlations for density, viscosity, heat capacity, and thermal conductivity. The user can select from built-in coolants, shown in Table 10 with references, or specify their own properties using the "Materials" input block (described in Section 3.2).

Table 10. DASSH built-in coolant properties to use in "coolant_material" input

Coolant	Reference
Sodium	[20]
Sodium-potassium eutectic	[21]
Lead	[22]
Bismuth	[22]
Lead-bismuth eutectic	[22]

The “gap_model” and “bypass_fraction” inputs describe the treatment of coolant heat transfer in the inter-assembly gap. Four gap models are available for users, as enumerated below.

- “flow”: inter-assembly coolant flows axially, with flow rate specified as a fraction of the total core flow rate by the “bypass_fraction”. Heat is transferred radially by conduction and axially by coolant flow. The low flow rates of the inter-assembly gap coolant can result in very small axial mesh sizes.

- “no_flow”: inter-assembly coolant is assumed to be “stagnant” such that heat is transferred between assemblies via conduction. There is no contact resistance between the coolant and the duct wall. Only radial heat transfer, via conduction, is considered between the active subchannel, the adjacent duct walls, and the adjacent coolant subchannels. Because coolant is not flowing, there is no axial mesh size requirement. This duct model introduces a small error in the core-wide energy balance because energy is not convected upward out of the inter-assembly gap.

- “duct_average”: the inter-assembly coolant temperature is calculated as the average of the adjacent duct walls. Only radial heat transfer is considered. This model is actually identical to the “no_flow” model except it neglects heat conduction between subchannels. There is no axial mesh size requirement. Like the no_flow model, this option also results in a small error in the core-wide energy balance because energy deposited in the coolant is not convected out from the inter-assembly gap.

- “none”: there is no inter-assembly heat transfer (adiabatic boundary at the outermost duct wall outer surface). Temperatures in the inter-assembly gap are not solved.

An example for a metallic-fueled sodium-cooled reactor is shown in Figure 9. The assembly pitch and core length are specified in meters. Built-in sodium properties are used for the coolant. The coolant inlet temperature is 648.15 K (375.0˚C). The inter-assembly gap heat transfer model is set to the "no_flow" model.

```
[Core]
    assembly_pitch       = 0.12            # m
    length               = 4.00            # m
    coolant_material     = sodium
    coolant_inlet_temp   = 648.15          # Kelvin (375.0 deg C)
    gap_model            = no_flow
```

Figure 9. Example input to the Core block

3.5 Assembly block

The Assembly input block describes the assembly geometry for each type of assembly loaded into the core. Each assembly type is described by a distinct subblock, for which all of the inputs are specified. The subblock name is the name of the assembly type; at least one assembly type must be provided. The required input is described in Table 11 below using the same conventions described above.

Table 11. Input arguments to the Assembly block

Keyword argument	Type	Req	Options and defaults	Description
Sub-block: [assembly name]				
num_rings	integer	X	---	Number of pin rings in assembly
pin_pitch	float	X	---	Pin-pin center pitch distance
pin_diameter	float	X	---	Pin outer diameter
wire_pitch	float	X	---	Wire wrap axial pitch; if no wire wrap, can equal 0.0
wire_diameter	float	X	---	Wire wrap diameter; if no wire wrap, can equal 0.0

clad_thickness	float	X	---	Cladding wall thickness
duct_ftf	list	X	---	Duct inner and outer flat-to-flat distances, from inside to outside ducts
duct_material	string	X	---	Material to use for duct properties
corr_mixing	bool	---	• MIT • CTD (default) • UCTD	Correlation to obtain mixing parameters: eddy diffusivity and swirl velocity
corr_friction	bool	---	• NOV • REH • ENG • CTD (default) • CTS • UCTD	Correlation to calculate bundle friction factor
corr_flowsplit	bool	---	• NOV • MIT • SE2 • CTD (default) • UCTD	Correlation to calculate flow split among coolant subchannels
htc_params_duct	list	---	default=0.025, 0.8, 0.8, 7.0	Dittus-Boelter correlation parameters to calculate Nu (see Equation 3-1)
bypass_gap_flow_fraction	float	---	default=0.05	For double-ducted assembly, coolant flow fraction to bypass gap between ducts
shape_factor	float	----	default=1.0	Factor to multiply the coolant thermal conductivity to affect coolant mixing.
use_low_fidelity_model	bool	---	default=False	Apply low-fidelity model to this assembly type; see Section 3.5.2
low_fidelity_model	string	---	• simple (default) • 6node	Choose the type of low-fidelity model to apply to this assembly; see Section 3.5.2

convection_factor	float, string	---	default=1.0; may equal "calculate" if specified for the entire assembly.	Weight applied to duct wall convection term in energy balance; see Section 3.5.2

For each assembly, the required inputs are the first set of keyword arguments that describe the fuel rod bundle geometry. The "duct_ftf" input requires the duct flat-to-flat distances, both inner and outer, be provided for each duct from the innermost duct to the outermost duct. The "duct_material" input points DASSH to the material properties for the duct wall. Users can choose from the built-in options, shown in Table 12, or can specify their own in the Materials input block (described in Section 3.2).

Table 12. Built-in structural material properties in DASSH

Material	Reference
HT9	[23]
D9	[24]
SS316	[25]
SS304	[26]

DASSH uses correlations for flow in wire-wrapped rod bundles to reduce the complexity of the problem. The user can specify the correlations used to describe coolant mixing, friction factor, and coolant flow velocity split in the rod bundle. The correlations and their references are shown in Table 13. It is recommended that the user become familiar with these references and assess how well their assembly geometry matches with the range of applicability. The Cheng-Todreas Detailed correlations are set as the defaults, as they are widely used, have a broad range of applicability, and were found to be the most accurate [27] [28].

Table 13. DASSH built-in coolant flow correlations

Correlation type	Correlation Name	Year	Reference
Friction factor	Novendstern (NOV)	1972	[29]
	Rehme (REH)	1973	[30]
	Engel (ENG)	1979	[31]
	Cheng-Todreas Simple (CTS)	1986	[32]
	Cheng-Todreas Detailed (CTD)	1986	[32]
	Upgraded Cheng-Todreas Detailed (UCTD)	2018	[33]
Flow split	Novendstern (NOV)	1972	[29]

	Chiu-Rohsenow-Todreas (MIT)	1980	[7]
	Cheng-Todreas Detailed (CTD)	1986	[34]
Coolant mixing	Chiu-Rohsenow-Todreas (MIT)	1980	[35]
	Cheng-Todreas (CTD)	1986	[34]

The Dittus-Boelter correlation, shown below, is the only option for the calculation of the Nusselt number, but the user can provide the coefficients to the equation using the "htc_params_duct" input. The Nusselt number is used to compute the heat transfer coefficient for convection between the coolant and duct wall. The default values for A, B, C, and D are 0.025, 0.8, 0.8, and 7.0, respectively, as given by the Lyon-Martinelli equation [36] [37] [38]; a review of other correlations for Nusselt number is given in Ref. [39] .

$$\mathrm{Nu} = A * (\mathrm{Re}^{B}\ \mathrm{Pr}^{C}) + D \qquad (3\text{-}1)$$

where: Nu = Nusselt number
Re = Reynolds number
Pr = Prandtl number
A, B, C, D = correlated coefficients

3.5.1 *Low-fidelity model regions*

DASSH offers a low-fidelity model to simplify calculations in axial regions that are not pin bundles. As an example, the design details of the input plenum or the lower or upper axial reflectors might not be known in the early design stages. Rather than treating these regions as an extension of the rod bundle (as done in SE2-ANL) or omitting them from the model altogether, the user can treat them as homogeneous mixtures of structure and coolant. A diagram of the low-fidelity model system is shown in Figure 10. The system has a single coolant node. Heat transfer takes place via forced axial convection and by convection with the duct wall, which occurs through six duct meshes centered at the corners. Power from neutron and gamma heating is added directly to the coolant at each axial step.

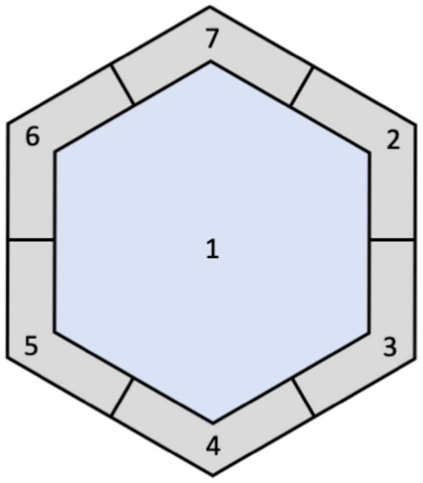


Figure 10. Coolant and duct subchannels for the low-fidelity model

The low-fidelity model can be applied to axial regions above and below the pin bundle by adding a sub-subblock titled "AxialRegion" to the desired assembly type. The arguments for the AxialRegion sub-subblock are given in Table 14.

Table 14. Input arguments to the AxialRegion sub-subblock of the Assembly block

Keyword argument	Type	Req	Options and defaults	Description
Sub-sub-subblock: [region name]				
z_lo	float	X	min=0.0	Region lower bound
z_hi	float	X	---	Region upper bound
vf_coolant	float	X	min=0.0, max=1.0	Coolant volume fraction
hydraulic_diameter	float	---	default=0.0	Hydraulic diameter of the axial region; used to calculate pressure drop
epsilon	list	---	default=0.0	Surface roughness; used to calculate friction factor
htc_params	list	---	default=0.025, 0.8, 0.8, 7.0	Dittus-Boelter correlation parameters to calculate Nu (see Equation 3-1)

The user must provide the axial boundaries of the region (keywords "z_lo" and "z_hi"). If desired, multiple homogeneous regions can be used within a given assembly. The rod bundle is assumed to be located in between lower and upper sets of un-rodded regions (for example, where "z_hi" does not equal the "z_lo" of another un-rodded region). The user must also specify the volume fraction of coolant in the region.

Many optional inputs are available for homogenized regions. If desired, the user can specify additional parameters used to calculate pressure drop in the region using the Darcy-Weisbach equation. These parameters are the hydraulic diameter and the surface roughness (epsilon). If not specified, the hydraulic diameter is calculated using an estimated wetted perimeter obtained by treating the non-coolant volume as a cylinder. The friction factor is calculated as shown below, where the equation for the turbulent friction factor is a hybrid of multiple correlations based on the Colebrook equation as presented in Equation 15 in [40]. The transition regime friction factor is a linear interpolation between the laminar and turbulent values, as shown in Equation 3.3-206 in [41].

$$\text{If } Re < 2200\text{:}\quad f_L(\mathrm{Re}) = \frac{64}{\mathrm{Re}} \tag{3-2}$$

$$\text{If } Re < 3000\text{:}\quad f_T(\mathrm{Re}) = 0.25\left[\log_{10}\left(\frac{\frac{\epsilon}{De}}{3.7} - \frac{4.518}{\mathrm{Re}}\log_{10}\left(\frac{6.9}{\mathrm{Re}} + \left[\frac{\frac{\epsilon}{De}}{3.7}\right]^{1.11}\right)\right)\right]^{-2} \tag{3-3}$$

$$\text{Otherwise:}\quad f_{Tr}(\mathrm{Re}) = f_L(2200) + \left[3.75 - \frac{8250.0}{\mathrm{Re}}\right]\left(f_T(3000) - f_L(2200)\right) \tag{3-4}$$

where: f_L = friction factor in the laminar regime
f_T = friction factor in the turbulent regime
ϵ = surface roughness
f_{Tr} = friction factor in the laminar-turbulent transition regime

The user can control the magnitude of convective heat transfer by modifying the coefficients used in the correlation for the Nusselt number, which is used to determine the heat transfer coefficient. Decreasing the Nusselt number decreases the heat transfer coefficient, which decreases the amount of heat transferred by convection to or from the duct wall.

3.5.2 *Application of low-fidelity model to assemblies*

Assemblies with very low power – that would therefore receive very low flow rates and impose harsh axial mesh size requirements on the problem – may be less important to treat in such detail. An example of such an assembly is a shield assembly at the periphery of a fast reactor core. The low-fidelity model in DASSH can be used to model these assemblies based on pin bundle specifications. By invoking the "use_low_fidelity_model" input, the assembly is treated as a homogeneous mixture of coolant and structural materials. The coolant volume fraction is calculated based on the pin bundle parameters. This treatment does not eliminate the axial mesh size requirement, but it will greatly relax it – generally by an order of magnitude or more.

When the low-fidelity model is used to model an assembly with pin bundle parameters as input, the heat transfer coefficient is calculated based on the correlated Nusselt number and the effective thermal conductivity of the fluid. The effective thermal conductivity is calculated using correlations to determine the eddy diffusivity, ε. That effective conductivity is also scaled by a lateral porosity factor λ that accounts for the presence of the pins, which impede lateral heat transfer through the bundle coolant.

$$h = \frac{\text{Nu}\, k^*}{De} = \frac{\text{Nu}\, \lambda (k s^* + \rho C_p \varepsilon)}{De} \tag{3-5}$$

where:

h	= coolant-to-wall heat transfer coefficient (W/m^{2}K)
k^*	= effective thermal conductivity (W/m-K)
De	= hydraulic diameter (m)
λ	= lateral porosity factor ($= 1 - D_{pin}/P_{pin}$)
s^*	= conductivity shape factor
ρ	= coolant density (kg/m^3)
C_p	= coolant heat capacity (J/kg-K)
ε	= correlated eddy diffusivity (m^2/s)

3.5.3 *Fuel pin temperature model*

If a user wants fuel pin temperatures to be calculated, they must provide inputs in the FuelModel sub-subblock, which are tabulated in Table 15. Fuel pin temperatures are calculated at each axial step of the DASSH coolant calculation.

Table 15. Input arguments to the FuelModel sub-subblock of the Assembly block

Keyword argument	Type	Req	Options and defaults	Description
Sub-subblock: FuelModel				
clad_material	string	X		Cladding material
gap_material	string	X*	default=None	Material in the fuel-clad gap
fcgap_thickness	float	---	default=0.0	Fuel-clad gap thickness
htc_params_clad	list	---	See Equation 3-5 for default	Dittus-Boelter correlation parameters to calculate Nu (see Equation 3-1)
r_frac	list	X	---	Radial fractions at which to divide fuel pellet, in descending order
pu_frac	list	X	---	Pu fraction in each radial shell described by r_frac
zr_frac	list	X	---	Zr fraction in each radial shell described by r_frac
porosity	list	X	---	Porosity in each radial shell described by r_frac
* Requirement is conditional on other inputs				

The clad material and clad thickness are mandatory inputs, whereas the gap material and gap thickness inputs are not; the default model is for the fuel to be in contact with the clad (in other words, gap thickness is zero). The keyword inputs "pu_frac", "zr_frac", and "porosity" characterize the radial distributions of Pu, Zr, and fuel porosity. These are specified in regions divided by fractional radii given in the "r_frac" input. Users should order these values from the outside in. A diagram of an example system with three regions is shown in Figure 11; the "r_frac" input that would describe this pin model is "r_frac = 0.67, 0.33, 0.0". If annular fuel, the last value in the "r_frac" input should be greater than zero.

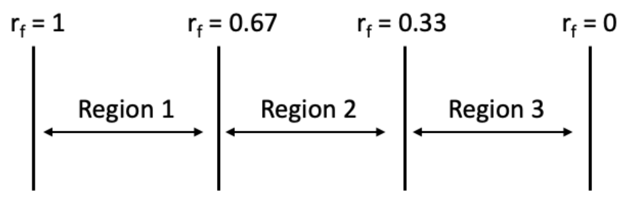


Figure 11. Diagram of example fuel pin region specifications

These values are required to calculate the fuel thermal conductivity, as shown in Sections C.1.2 and C.1.3 in Ref. [24]. Currently the only model for fuel thermal conductivity is for metal fuels, as is the case in SE2-ANL. In the future, additional models will be added to reflect the user's specifications for fuel heating. The user can input coefficients to the Dittus-Boelter correlation for Nusselt number for the heat transfer coefficient between coolant to cladding using the "htc_params_clad" input. As a default, these are set to be calculated based on pin geometry parameters, as shown below.

$$A = \frac{1}{3}\left(\frac{P}{D}\right)^{3.8} 0.01^{0.86} \qquad B = C = 0.86 \qquad D = 4.0 + 0.16\left(\frac{P}{D}\right)^{5} \qquad (3\text{-}6)$$

where: P = pin pitch (m)
D = pin diameter (m)

An example of an Assembly input block featuring AxialRegion and FuelModel sub-subblocks is shown in Figure 12. This assembly is the metallic driver fuel assembly designed for the ABR-1000 reactor [42]. The input takes on the default correlations and Nusselt number correlation parameters and omits the pressure drop calculation in the low-fidelity model region. The fuel model assumes no gap between the fuel and clad.

```
[Assembly]
    [[fuel]]
        num_rings     = 10                    # 271 pins
        pin_pitch     = 0.008909              # m
        pin_diameter  = 0.007550              # m
        wire_pitch    = 0.204190              # m
        wire_diameter = 0.00131               # m
        duct_ftf      = 0.14922, 0.15710      # m; flat-to-flat from inside-out
        duct_material = HT9

        [[[AxialRegion]]]
            # Model un-fueled region as un-rodded
            [[[[nosepiece]]]]
                z_lo = 0.0           # m
                z_hi = 0.3556        # m
                vf_coolant = 0.10
            [[[[lower_shield]]]]
                z_lo = 0.3556        # m
                z_hi = 1.6002        # m
                vf_coolant = 0.25
            # Fueled region is 1.6002 m – 2.4130 m
            # Gas plenum is 2.4130 m – 3.6576 m
            [[[[upper_components]]]]
                z_lo = 3.6576        # m
                z_hi = 4.7752        # m
                vf_coolant = 0.25

        [[[FuelModel]]]
            clad_thickness = 0.00056 # m
            clad_material  = HT9
            r_frac   =  0.0, 0.33333, 0.66667     # Non-annular fuel
            pu_frac  = 0.20,    0.20,    0.20     # Constant Pu dist.
            zr_frac  = 0.10,    0.10,    0.10     # Constant Zr dist.
            porosity = 0.25,    0.25,    0.25     # Constant porosity dist.
```

Figure 12. Example input to the Assembly block

3.6 Assignment Block

The Assignment input block is where users connect assemblies, as defined in the Assembly block, to positions in the core. It is also where users specify boundary conditions – either a flow rate or a temperature constraint – for each assembly. The Assignment block breaks from the keyword-value pair format used throughout the rest of the DASSH input and is processed outside ConfigObj. Users familiar with DIF3D may recognize that it mimics the format used in Card Type 30 in the DIF3D A.NIP3 input. The format is shown in the following code block.

```
[Assignment]
    [[ByPosition]]
        name = ring, start_position, end_position, boundary_condition=value
        # types: string = int, int, int, string = float
        # …
        # repeat until every core location is filled
```

Figure 13. Input description to the Assignment block

To further illustrate the usage, an example is shown below that demonstrates the position assignment conventions and the options for boundary conditions.

```
[Assignment]
    [[ByPosition]]
        fuel = 1,  1,  1, delta_temp=150   # ring 1, position 1;     BC1
        fuel = 2,  1,  6, outlet_temp=770  # ring 1, positions 1-6;  BC2
        refl = 3,  1,  5, flowrate=30      # ring 3, positions 1-5;  BC3
        refl = 3,  6,  6, delta_temp=150   # ring 3, position 6;     BC1
        refl = 3,  7, 11, flowrate=30      # ring 3, positions 7-11; BC3
        refl = 3, 12, 12, delta_temp=150   # ring 3, position 12;    BC1

# BC1: temperature increase across core (Kelvin)
# BC2: coolant outlet temperature (Kelvin)
# BC3: coolant flow rate (kg/s)
```

Figure 14. Example input to the Assignment block

In the above, an assembly named “fuel” (as specified in the Assembly block) is loaded into the center position (“1, 1, 1” indicates “ring 1, start position 1, end position 1”) and is assigned the boundary condition that the coolant temperature rise across it be equal to 150 degrees K. The assembly type “fuel” is also assigned to every position in the second ring (“2, 1, 6” indicates “ring 2, start position1, end position 6” and it given the boundary condition that the outlet coolant temperature equals 770 degrees K. DASSH will estimate the flow to assign to these assemblies using $Q = mC_p dT$ and neglects consideration of inter-assembly heat transfer.

A second assembly type, “refl”, is assigned to the third ring. Two positions on that ring have flow rate boundary conditions, with units of kg/s. Important to note here is that the overlay feature of DIF3D input (an entire ring can be defined on one line and individual positions are "overwritten" afterward) is not implemented in DASSH at this time. It is necessary that users individually specify conditions at every possible assembly position.

3.7 Plot block: visualization in DASSH

DASSH offers built-in visualization post-processing capabilities using the mainstream Python plotting package matplotlib [17]. DASSH accepts user requests for figures through the Plot input block. DASSH generates figures based on data dumped to CSV files during the sweep, which the user can control from the Dump subblock of the Setup input block. If the Plot input is present and the user requests no data to be dumped, DASSH will automatically dump all the necessary data to make the figures.

General arguments for the Plot input block are shown in Table 16. Depending on the figure type requested, different input arguments are activated (or inactivated). In the following sub-subsections, the input required to produce each type of figure is described separately.

Table 16. DASSH Plot input arguments

Keyword argument	Type	Req	Options and defaults	Description
Sub-block: [plot name]				
type	integer	X	• SubchannelPlot • PinPlot • CoreSubchannelPlot • CorePinPlot • CoreHexPlot	Type of figure
z	float (list)	X*	---	Axial height to plot data (from linear interpolation between solution planes)
assembly_id	integer (list)	X*	---	Assembly for which to plot values
value	string (list)	X*	---	Value to plot, if necessary
cmap	string	---	default: jet	Color map to use (see [43])
cbar_lbnd	float	---	default: data minimum	Colorbar lower bound

cbar_mpnt	float	---	default: average of data minimum and maximum	Colorbar midpoint
cbar_ubnd	float	---	default: data maximum	Colorbar upper bound
cbar_label	string	---	default=Temperature ([unit])	Colorbar label
dpi	float	---	default: 200	Figure (.png) DPI specification
units	string	---	default: SI units	Units with which to plot data in the figure
rings	integer	---	default=None	Number of assembly rings to plot in core-wide plots
ignore_simple	bool	---	default=False	Treatment of un-rodded regions and assemblies in CoreSubchannelPlot
pins	bool	---	default=False	Overlay circles on pin positions in SubchannelPlot
pin_alpha	float	---	default=1.0 (fully opaque)	Opacity of pin circles in SubchannelPlot
data_label	bool	---	default=True	Add data labels to CoreHexPlot
omit_nonvalue_rings	bool	---	default=False	Ignore outer rings if no values to plot in CoreHexPlot
* Requirement is conditional on other inputs				

In general, DASSH gives users a great degree of control over figure generation. For all figure types, users can specify: the temperature units to use in plotting, if not SI units; the colormap to use for the colorbar; the lower bound, midpoint, and upper bound for the colorbar; and the DPI with which to save the figures (as .png files). Specific plot types have additional user options, as shown in Table 16 and illustrated in the specific examples that follow.

3.7.1 SubchannelPlot: assembly subchannel temperatures

Users can plot subchannel temperatures in individual assemblies using the "SubchannelPlot" plot type. For this type of figure, the user must specify the assembly ID to plot and the axial height at which the data should be plotted. The assembly ID corresponds to the indexing option selected in the Options subblock of the Setup input block. Users may provide a list of comma-separated values for these inputs to produce combinations of figures. An example is shown in Figure 15, along with an example input that would generate it in Figure 16.

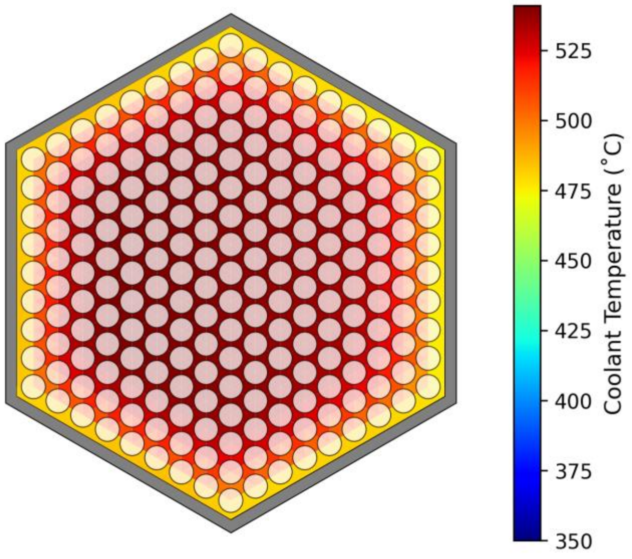


Figure 15. Example SubchannelPlot figure with pins overlaid

```
[Plot]
    [[example_1]]
        type        = SubchannelPlot
        assembly_id = 1       # request multiple with 1, 2, 3, …
        z           = 2.00    # m; request multiple with 2.00, 2.10, …
        dpi         = 200     # default
        units       = Celsius # Add if different than default (K)
        cbar_label  = Coolant Temperature (˚C)
        cmap        = jet     # default
        cbar_lbnd   = 350.0   # Celsius; request alternate from default
        cbar_mpnt   = None    # default; let DASSH decide
        cbar_ubnd   = None    # default; let DASSH decide
        pins        = True    # overlay pin positions
        pin_alpha   = 0.75    # opacity of pin circles (1.0 is fully opaque)
```

Figure 16. Example SubchannelPlot input showing all relevant input arguments

3.7.2 PinPlot: assembly pin temperatures

Users can plot pin temperatures in individual assemblies using the "PinPlot" plot type. For this type of figure, the user must specify the assembly ID to plot and the axial height at which the data should be plotted. Users may provide a list of comma-separated values for these inputs to produce combinations of figures. The user must also specify the type of data to plot. The available options are:

- clad_od: clad outer temperature
- clad_mw: clad mid-wall temperature,
- clad_id: clad inner temperature
- fuel_od: fuel outer temperature
- fuel_cl: fuel centerline temperature

Figure 17 shows an example of the PinPlot type and Figure 18 shows the input that can be used to generate it, including all possible arguments for this plot type.

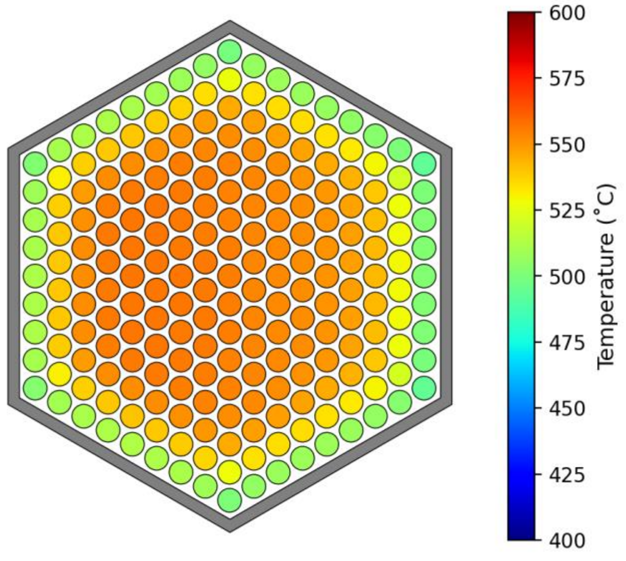


Figure 17. Example PinPlot figure

```
[Plot]
    [[example_2]]
        type        = PinPlot
        value       = clad_mw
        assembly_id = 1       # request multiple with 1, 2, 3, …
        z           = 2.0     # m; request multiple with 2.0, 2.1, …
        dpi = 200             # default
        units       = Celsius # Add if different from default units (K)
        cbar_label  = Temperature (˚C)
        cmap        = jet     # default
        cbar_lbnd   = 400.0   # Celsius
        cbar_mpnt   = None    # default; let DASSH decide
        cbar_ubnd   = 600.0   # Celsius
```

Figure 18. Example PinPlot input showing all relevant input arguments

3.7.3 CoreSubchannelPlot: core-wide subchannel temperatures

Users can extend the SubchannelPlot figure, as described and exemplified in Section 3.7.1, to all assemblies in the core using the CoreSubchannelPlot type. The inputs are the same as for the SubchannelPlot except that no assembly ID is required and there is no capability to overlay pins on the figure. Additionally, the user can limit the number of assembly rings plotted in the figure with the "rings" argument (rings=1 would plot only the central assembly).

Depending on the problem setup, there may be low-fidelity model regions and assemblies at the height requested in the CoreSubchannelPlot input. DASSH gives the user the ability to control how these assemblies are plotted with the "ignore_simple" keyword argument. If True, assemblies modeled with the low-fidelity model are omitted from the figure and plotted with a gray hexagon. If False (default), they are plotted with a hexagon whose color corresponds to the bulk coolant temperature in that assembly at that height, according to the colorbar in the figure.

Figure 19 and Figure 20 show an example CoreSubchannelPlot for a 19-assembly core and the input used to generate it, respectively.

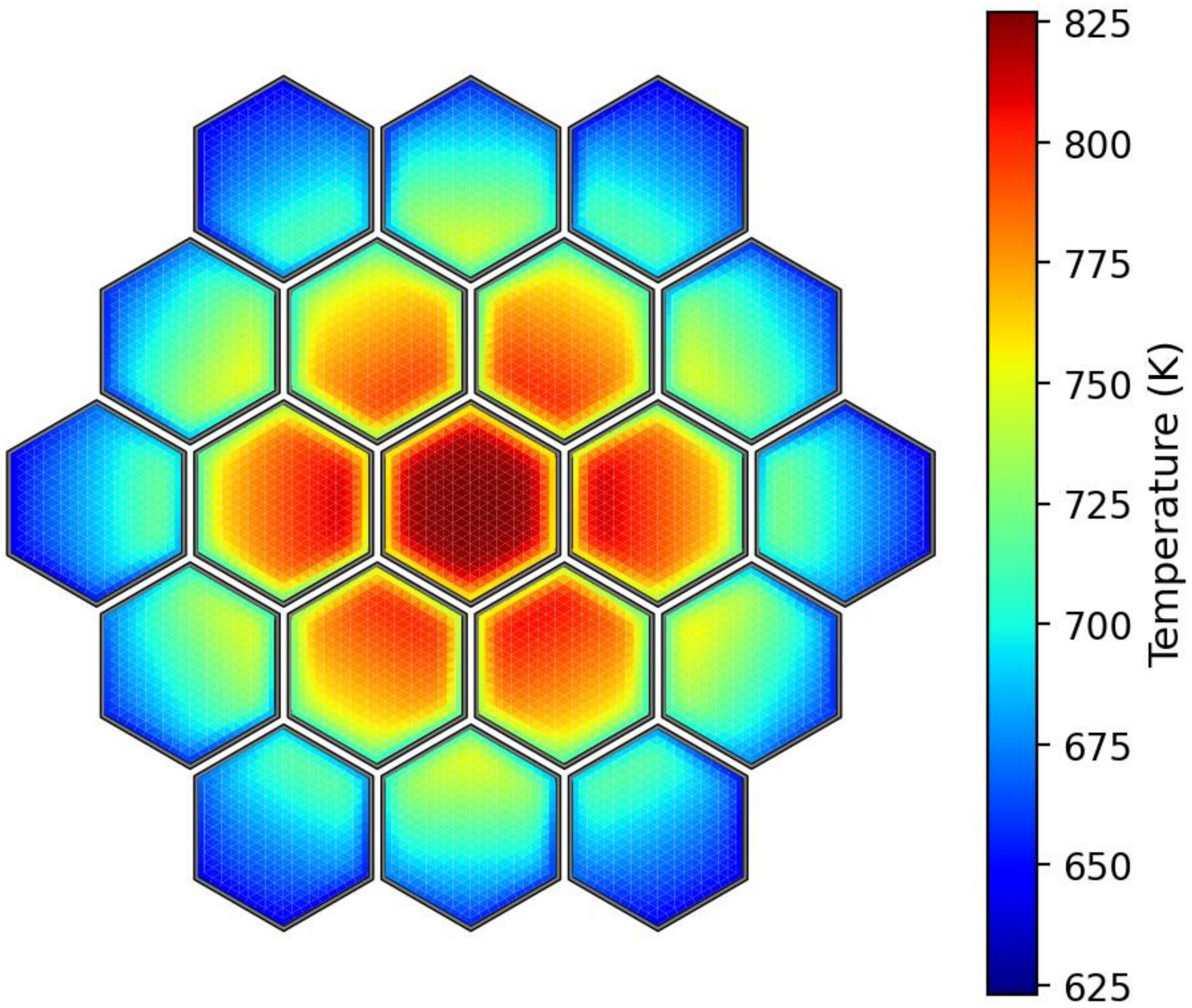


Figure 19. Example CoreSubchannelPlot figure

```
[Plot]
    [[example_3]]
        type       = CoreSubchannelPlot
        z          = 2.5             # m; input multiple w/ 2.0, 2.1, …
        dpi        = 200             # default
        cmap       = None            # DASSH uses default=jet
        units      = Kelvin          # default
        cbar_label = Temperature (K)
        cbar_lbnd  = None            # default; let DASSH decide
        cbar_mpnt  = None            # default; let DASSH decide
        cbar_ubnd  = None            # default; let DASSH decide
        rings      = 3               # only three rings are plotted
        ignore_simple = False        # default
```

Figure 20. Example PinPlot input showing all relevant input arguments

3.7.4 CorePinPlot: core-wide pin temperatures

Users can extend the PinPlot figure, as described and exemplified in Section 3.7.2,, to all assemblies in the core using the CorePinPlot type. The inputs are the same as for PinPlot except that no assembly ID is required. Like for the CoreSubchannelPlot type, the user can specify the number of assembly rings to be included in the figure. Low-fidelity model regions and assemblies are automatically ignored because they have no pin temperatures to visualize. Figure 19 and Figure 20 show an example CorePinPlot for a 19-assembly core and the input used to generate it, respectively.

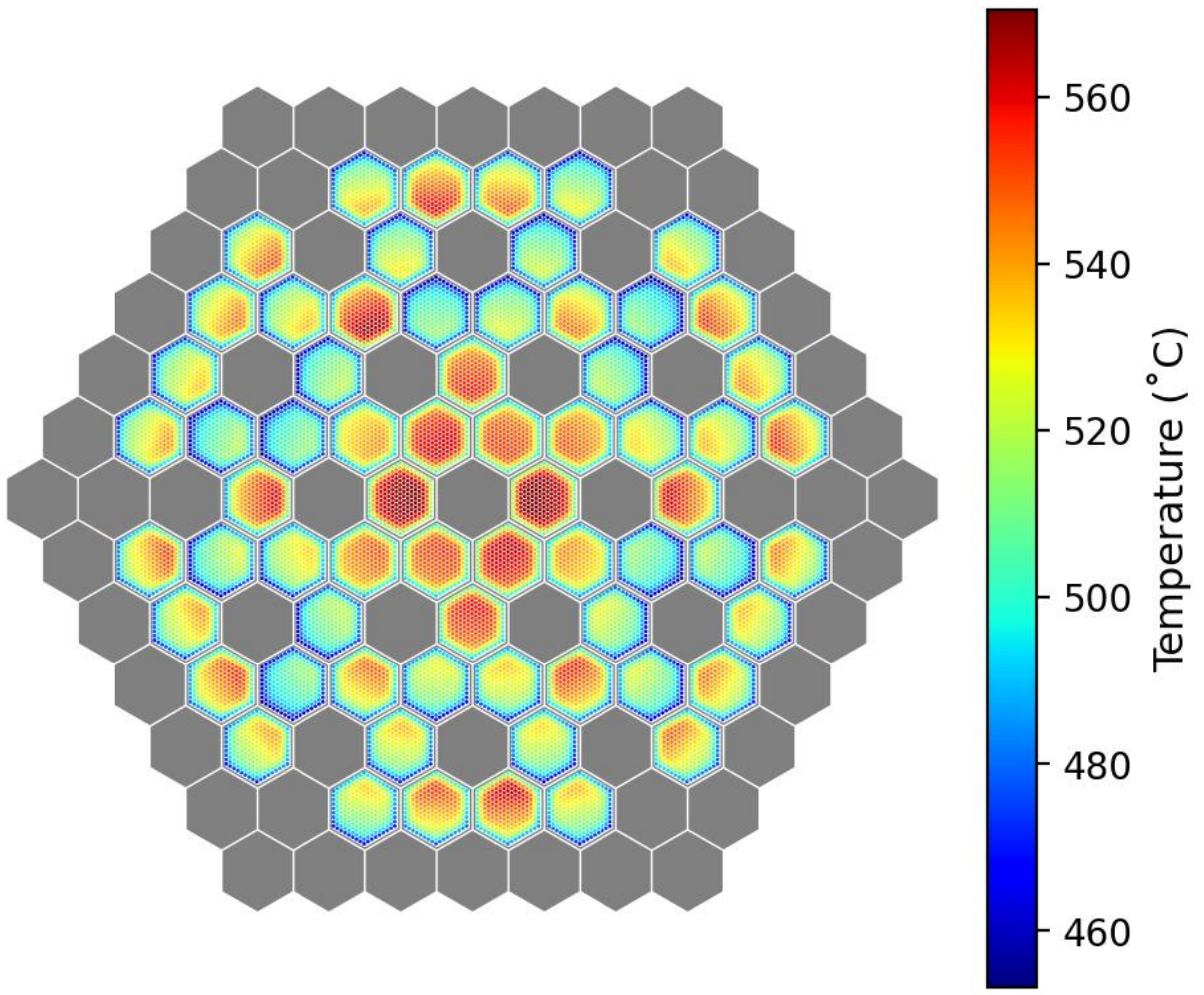


Figure 21. Example CorePinPlot

```
[Plot]
    [[example_4]]
        type       = CorePinPlot
        value      = clad_mw
        z          = 2.0      # m; request multiple with 2.0, 2.1, …
        dpi        = 200      # default
        units      = Celsius  # add if different than default (K)
        cbar_label = Temperature (˚C)
        cmap       = None     # default = jet
        cbar_lbnd  = None     # default; let DASSH decide
        cbar_mpnt  = None     # default; let DASSH decide
        cbar_ubnd  = None     # default; let DASSH decide
        rings      = 7        # include 7 hex rings
```

Figure 22. Example CorePinPlot input showing all relevant input arguments

3.7.5 CoreHexPlot: core-wide plot of values in hexagonal grid

Users can plot values on a hexagonal grid using the CoreHexPlot type. Multiple values can be visualized with this plot type, and the arguments required vary accordingly. All CoreHexPlot figures require input to the “value” keyword argument. Depending on that input and the type of data to be plotted, the user may also need to specify an axial height with the “z” keyword argument. These combinations are summarized in Table 17 and explained below.

Table 17. Input combinations for CoreHexPlot

Value	Keyword	Axial position required
Total power	total_power	N
Axial-maximum temperatures	• max_coolant_temp • max_duct_mw_temp • max_clad_mw_temp • max_fuel_cl_temp	N
Radial-maximum temperatures	• max_coolant_temp • max_duct_mw_temp • max_clad_mw_temp • max_fuel_cl_temp	Y
Radial-average temperatures	• avg_coolant_temp • avg_duct_mw_temp • avg_clad_mw_temp • avg_fuel_cl_temp	Y

To generate a figure plotting the total power in each assembly, as calculated by DASSH during the sweep, the value should be set to "total_power". No axial height is required because the value reflects a total accumulated over the entire length of the core.

To generate figures plotting overall maximum temperatures, the value should be set to one of the options indicated in Table 17 for axial-maximum temperatures. No axial height should be provided because the value should reflect the maximum over the entire length of the core.

By contrast, to generate figures that show radial maximum or average temperatures at a specific axial height, the value should be set to one of the options indicated in the third and fourth rows of Table 17 and an axial height should be specified using the "z" keyword argument.

The CoreHexPlot type two features additional plot customization options beyond those generally available to all the figures. First, because fuel assemblies are generally located near the core center, pin temperatures in non-fuel assemblies away from the core center may not be modeled. This potentially creates many non-data assemblies to ignore when plotting average or maximum clad mid-wall or fuel centerline temperatures. Therefore, the "omit_nonvalue_rings" argument enables the omission of all outer rings that contain no values to plot. In other words, if this argument is set True, once all the assemblies with values to plot have been added to the figure, the rest will be omitted. Second, because the CoreHexPlot shows only a single value in each hex, data labels can be easily added without overcrowding the figure. Setting the "data_label" input argument to True results in the inclusion of data labels that overlay the hexes.

An example CoreHexPlot, which features the use of both the "omit_nonvalue_rings" and "data_label" input arguments, is shown in Figure 23. The input that generated this figure is shown in Figure 24.

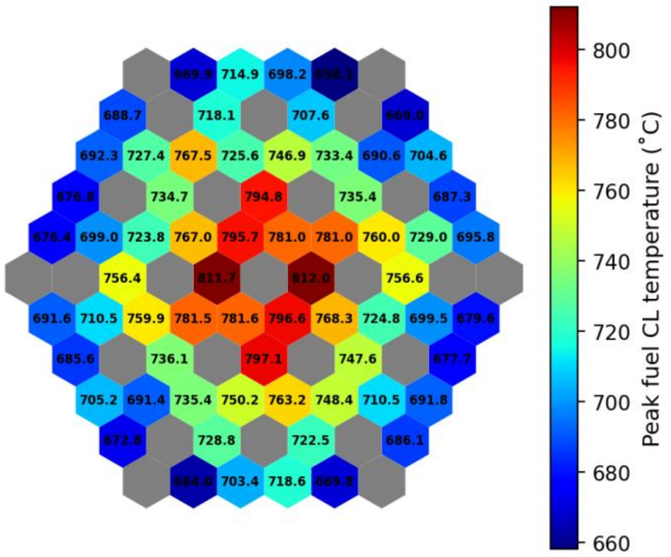


Figure 23. Example CoreHexPlot

```
[Plot]
    [[example_5]]
        type       = CoreHexPlot
        value      = fuel_cl
        z          = 1.75     # m; request multiple with 2.0, 2.1, …
        dpi        = 200      # default
        units      = Celsius  # add if different than default (K)
        cbar_label = Temperature (˚C)
        cmap       = None     # default = jet
        cbar_lbnd  = None     # default; let DASSH decide
        cbar_mpnt  = None     # default; let DASSH decide
        cbar_ubnd  = None     # default; let DASSH decide
        data_label = True
        omit_nonvalue_rings = True
```

Figure 24. Example CoreHexPlot input showing all relevant input arguments

4 Executing DASSH

Once an input has been prepared, a user is ready to execute DASSH. This is accomplished from the command line by invocating the command "dassh". The first argument must be a path to the input file for the problem at hand, as shown below. After that, multiple input flags are available for users to control the DASSH execution.

```
dassh path/to/input/file [--options]
```

The option flags, invoked with a double-dash and then a keyword, toggle certain runtime environment conditions. These are described below in Table 18.

Table 18. DASSH command line user options

Option	Description
--profile	Profile the cumulative runtime of each method in DASSH; produces a binary output file "dassh_profile.out" that can be processed in Python using the "pstats" package.
--save_reactor	Save the DASSH "Reactor" object, which contains characteristics of the core and assemblies (saved as "dassh_reactor.pkl").

The DASSH execution procedure proceeds as follows:

1. The user input is read and checked for consistency. Any errors regarding user-specified parameters should be raised early.
2. The power distribution is calculated.
3. The assembly pins and subchannels and the inter-assembly gap subchannels are mapped and their adjacency recorded. Boundary conditions for each assembly are processed, resulting in the assignment of a coolant flow rate to each assembly.
4. The first half of the summary output file is generated. This is created early to give the user early feedback on their problem. Information about the axial step size requirement imposed by each assembly may be useful for users to change the problem set up if the execution time is expected to be exceedingly long. Note that

DASSH will produce a warning if the step size is less than 0.5 mm or there are more than 2500 axial steps required.

5. DASSH performs the temperature sweep, calculating coolant, duct, and pin temperatures, if requested, throughout the core.
6. The summary output is completed, reporting the results of the temperature sweep.

Throughout the DASSH execution, some output is printed to the screen to report on progress through the problem. Additionally, a log file tracks the execution and reports any warnings or errors.

4.1 Visualization

As discussed in the previous section, DASSH features built-in capabilities to visualize results. Visualization can be invoked in two ways. First, plots may be requested in the main DASSH input file along with the rest of the input provided to DASSH when it is initially called. After performing the solution sweep and generating output, DASSH will automatically postprocess the dumped data to generate the requested plots.

Second, if a DASSH calculation has already been performed and the user wishes to generate new or different plots with existing data, they may do so with the "dassh_plot" command, as shown below. This invokes only the DASSH postprocessing step and leverages the data generated in a prior sweep. The command line invocation is shown below.

```
dassh_plot path/to/dassh_plot/input/file
```

Plotting after the DASSH calculation requires that the necessary CSV files and the binary DASSH data object "dassh_reactor.pkl" be requested during the calculation. The CSVs are requested using the Dump subblock of the Setup block; the dassh_reactor.pkl file is requested with the "--save_reactor" input option described above in Table 18.

4.2 Troubleshooting

A rule of thumb for troubleshooting errors in DASSH: if the error is reported in a logged error message, it was anticipated and likely occurs due to your input or due to a problem-specific failure in DASSH. Many of the error messages – including all that may be raised during input-checking – point toward the source of the error and explain why it arose. If the error is reported via a Python exception traceback, it was unanticipated and may indicate a bug in DASSH. Either way, users should feel free to reach out to the developers for help debugging and addressing issues.

5 Output description

DASSH produces two main types of output files. The primary DASSH output summarizes the modeled conditions and results in a text file. If requested by the user in the input file, DASSH will dump user-requested temperature data to CSV files. This section describes the format of these output files. In addition, the binary file containing the final DASSH system state is briefly introduced.

5.1 Summary output: dassh.out

The primary DASSH output is the summary output, recorded in a file called "dassh.out". DASSH always generates this output file. There, multiple tables provide an overview of temperatures throughout the core. For most problems, this output will be sufficient. There are 6 sections, described below in Table 19.

Table 19. Description of summary output file sections

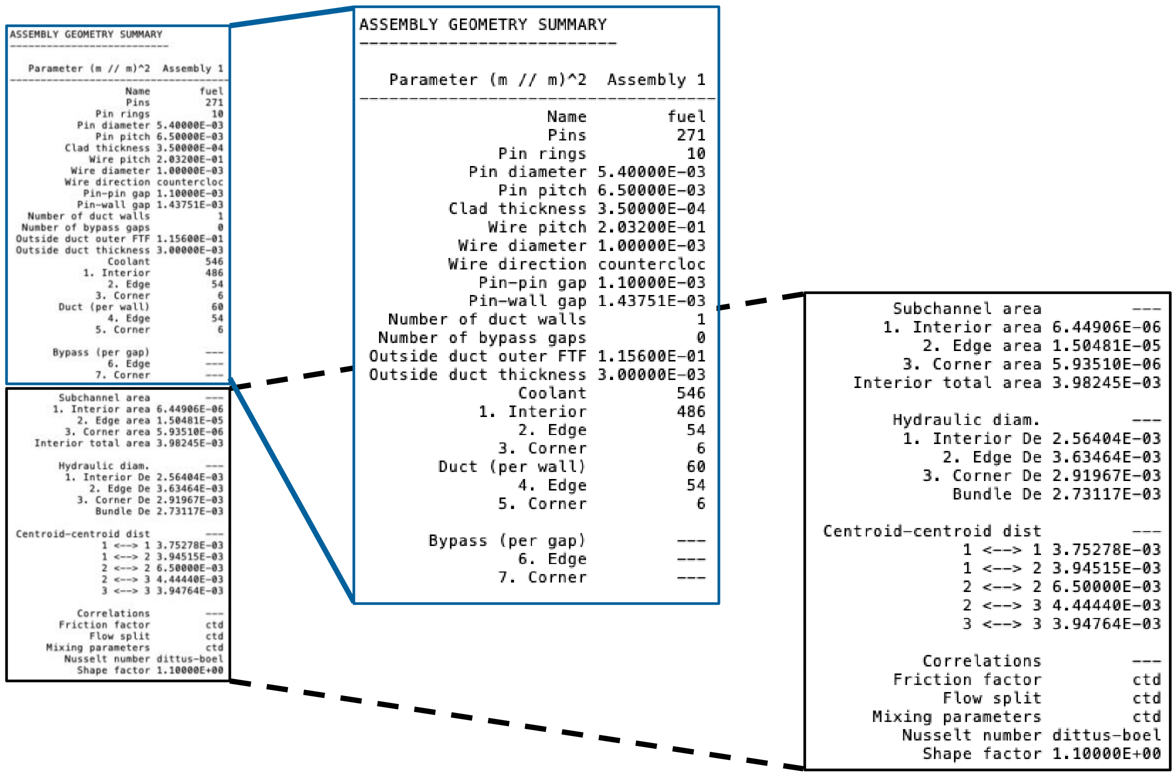
Num.	Name	Description
1	Assembly geometry summary	Overview of each assembly type defined in the input; includes characteristics of the subchannels as well as the assigned correlations.
2	Assembly power and assigned flow rate	Lists the position, total power, and total flow rate for each assembly in the core. Additionally, reports the axial step size constraint and indicates the constraining subchannel.
3	Subchannel flow characteristics	Reports average and individual subchannel velocities and correlated parameters for each assembly in the core based on the inlet conditions. Note: these values will change throughout the sweep as coolant properties change with increasing temperature.
4	Pressure drop across assemblies	Reports pressure drop across each axial region, as well as the total pressure drop across the entire assembly.
5	Overall assembly energy balance	Reports coolant energy balances for each assembly in the core; populated depending on user input.
6	Inter-assembly energy balance	Reports energy transferred through the inter-assembly gap coolant arranged by duct hex face for each assembly in the core; populated depending on user input.
7	Coolant temperature summary	Lists coolant average and peak outlet temperatures, as well as the overall peak coolant temperature and the axial height at which it occurs.

8	Duct temperature summary	Lists average duct temperature at the assembly outlet for each hex face, as well as the overall peak duct temperature and the axial height at which it occurs.
9	Peak pin temperatures	The radial temperature profiles in the pins with the peak clad mid-wall and peak fuel centerline temperatures are reported (at the height at which those peak temperatures occur).

DASSH writes the first three tables in the output file at the start of the calculation. This gives users the opportunity to study Table 2 (which reports the axial mesh size constraint and limiting subchannel for each assembly) early in the calculation. This may inform modifications to the assembly geometry, boundary conditions, or assumptions need to be made to accelerate the calculation. The remaining tables in the output file are written at the end of the calculation, after the axial sweep. These summary results provide general design-basis information such as average and peak coolant and duct temperatures and radial pin temperature profiles at the locations of peak clad and peak fuel temperature.

5.1.1 Table 1: Pin bundle geometry

An example of Table 1 is shown in Figure 25. It reports assembly pin bundle geometry parameters in columns for each assembly specified by the user. In addition to summarizing the user input, it displays the number of subchannels, their area, hydraulic diameter, and the distance between them. The correlations used in the calculation are displayed as well.

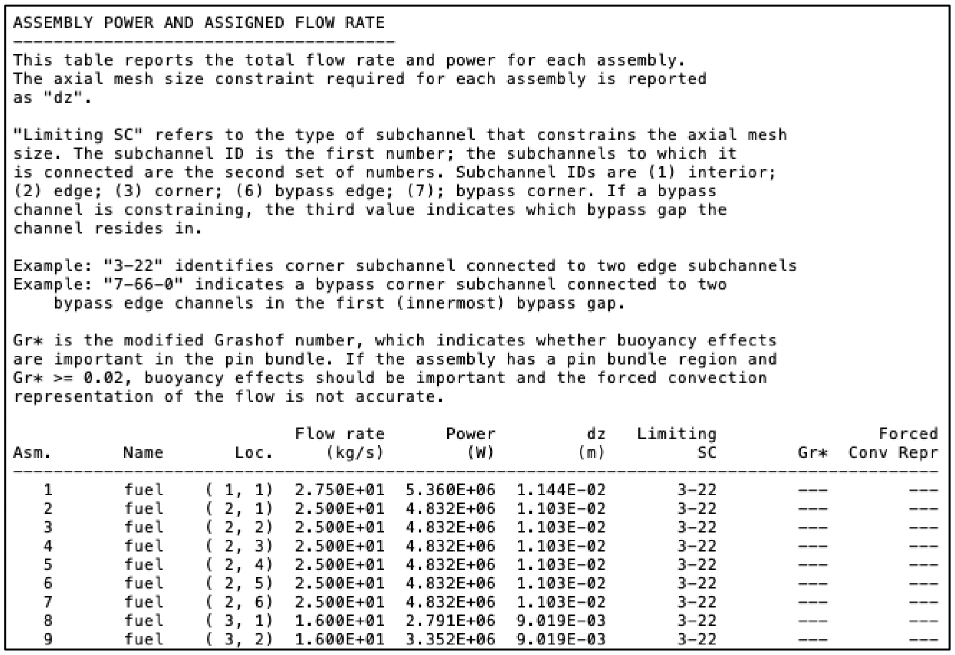


```
ASSEMBLY GEOMETRY SUMMARY
-------------------------

  Parameter (m // m)^2  Assembly 1
----------------------------------------
                  Name         fuel
                  Pins          271
             Pin rings           10
          Pin diameter  5.40000E-03
             Pin pitch  6.50000E-03
        Clad thickness  3.50000E-04
            Wire pitch  2.03200E-01
         Wire diameter  1.00000E-03
        Wire direction  countercloc
           Pin-pin gap  1.10000E-03
          Pin-wall gap  1.43751E-03
   Number of duct walls           1
  Number of bypass gaps           0
 Outside duct outer FTF  1.15600E-01
 Outside duct thickness  3.00000E-03
               Coolant          546
           1. Interior          486
               2. Edge           54
             3. Corner            6
       Duct (per wall)           60
               4. Edge           54
             5. Corner            6

        Bypass (per gap)         ---
               6. Edge          ---
             7. Corner          ---

        Subchannel area          ---
     1. Interior area  6.44906E-06
         2. Edge area  1.50481E-05
       3. Corner area  5.93510E-06
  Interior total area  3.98245E-03

       Hydraulic diam.          ---
       1. Interior De  2.56404E-03
           2. Edge De  3.63464E-03
         3. Corner De  2.91967E-03
            Bundle De  2.73117E-03

Centroid-centroid dist          ---
            1 <--> 1  3.75278E-03
            1 <--> 2  3.94515E-03
            2 <--> 2  6.50000E-03
            2 <--> 3  4.44440E-03
            3 <--> 3  3.94764E-03

         Correlations          ---
       Friction factor          ctd
            Flow split          ctd
     Mixing parameters          ctd
        Nusselt number  dittus-boel
          Shape factor  1.10000E+00
```

Figure 25. Example pin bundle geometry table from summary output

5.1.2 Table 2: Assembly assigned power and flow rate

Table 2 reports the power and flow rate assigned to each assembly in the core, to confirm user-input boundary conditions. It also reports the axial mesh size constraint for that assembly and the subchannel for which the mesh size constraint is obtained. Finally, Table 2 also reports the critical Grashof criteria for the bundle, which indicates whether the forced convection assumption is appropriate or whether natural convection will be present. An example is shown in Figure 26; there, the critical Grashof criteria is not calculated because the coolant was assigned constant material properties and therefore has no thermal volumetric expansion.

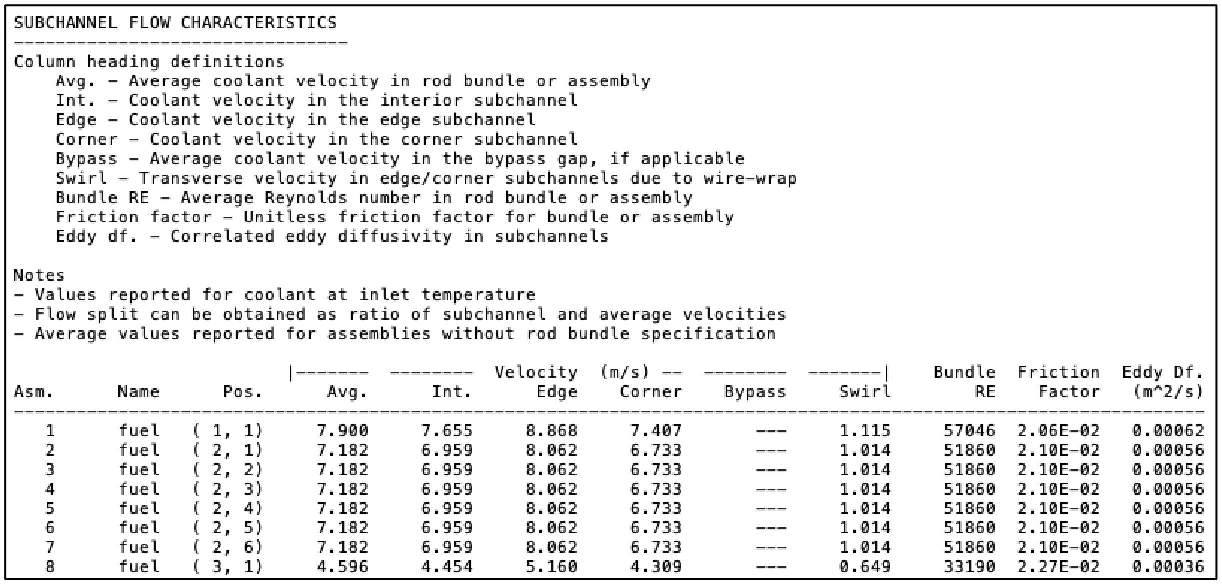


```
ASSEMBLY POWER AND ASSIGNED FLOW RATE
--------------------------------------
This table reports the total flow rate and power for each assembly.
The axial mesh size constraint required for each assembly is reported
as "dz".

"Limiting SC" refers to the type of subchannel that constrains the axial mesh
size. The subchannel ID is the first number; the subchannels to which it
is connected are the second set of numbers. Subchannel IDs are (1) interior;
(2) edge; (3) corner; (6) bypass edge; (7); bypass corner. If a bypass
channel is constraining, the third value indicates which bypass gap the
channel resides in.

Example: "3-22" identifies corner subchannel connected to two edge subchannels
Example: "7-66-0" indicates a bypass corner subchannel connected to two
    bypass edge channels in the first (innermost) bypass gap.

Gr* is the modified Grashof number, which indicates whether buoyancy effects
are important in the pin bundle. If the assembly has a pin bundle region and
Gr* >= 0.02, buoyancy effects should be important and the forced convection
representation of the flow is not accurate.

                              Flow rate      Power         dz   Limiting                Forced
Asm.        Name        Loc.     (kg/s)        (W)        (m)         SC        Gr*  Conv Repr
----------------------------------------------------------------------------------------------
   1        fuel     ( 1, 1)  2.750E+01  5.360E+06  1.144E-02       3-22        ---        ---
   2        fuel     ( 2, 1)  2.500E+01  4.832E+06  1.103E-02       3-22        ---        ---
   3        fuel     ( 2, 2)  2.500E+01  4.832E+06  1.103E-02       3-22        ---        ---
   4        fuel     ( 2, 3)  2.500E+01  4.832E+06  1.103E-02       3-22        ---        ---
   5        fuel     ( 2, 4)  2.500E+01  4.832E+06  1.103E-02       3-22        ---        ---
   6        fuel     ( 2, 5)  2.500E+01  4.832E+06  1.103E-02       3-22        ---        ---
   7        fuel     ( 2, 6)  2.500E+01  4.832E+06  1.103E-02       3-22        ---        ---
   8        fuel     ( 3, 1)  1.600E+01  2.791E+06  9.019E-03       3-22        ---        ---
   9        fuel     ( 3, 2)  1.600E+01  3.352E+06  9.019E-03       3-22        ---        ---
```

Figure 26. Example assembly assignment table from summary output

5.1.3 Table 3: Subchannel velocity and flow characteristics

Table 3 includes subchannel flow characteristics, including bundle-average, interior-type, edge-type, and corner-type velocities. The values obtained by the flow split correlation can be obtained by taking the ratio of the subchannel velocity and the bundle-average velocity (e.g., interior subchannel flow split is the ratio of the interior subchannel velocity and the bundle-average velocity). It also reports on other correlated parameters such as the swirl velocity, friction factor, and eddy diffusivity. An example is shown in Figure 27.

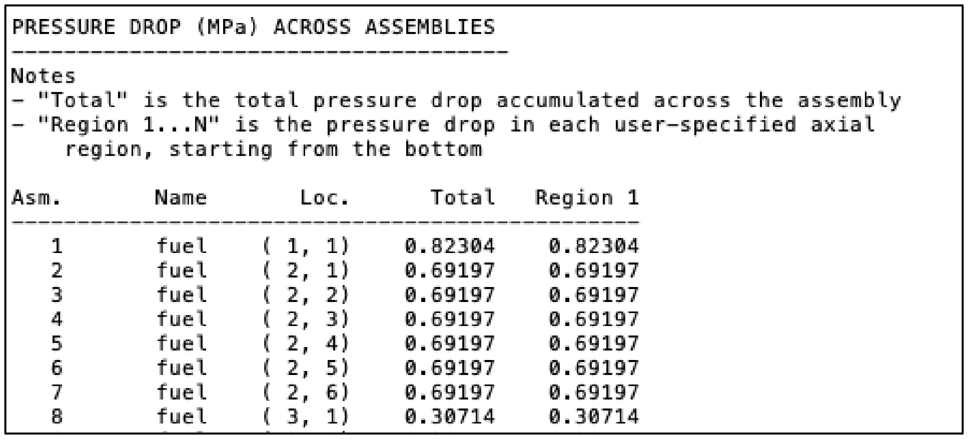


```
SUBCHANNEL FLOW CHARACTERISTICS
--------------------------------
Column heading definitions
    Avg. - Average coolant velocity in rod bundle or assembly
    Int. - Coolant velocity in the interior subchannel
    Edge - Coolant velocity in the edge subchannel
    Corner - Coolant velocity in the corner subchannel
    Bypass - Average coolant velocity in the bypass gap, if applicable
    Swirl - Transverse velocity in edge/corner subchannels due to wire-wrap
    Bundle RE - Average Reynolds number in rod bundle or assembly
    Friction factor - Unitless friction factor for bundle or assembly
    Eddy df. - Correlated eddy diffusivity in subchannels

Notes
- Values reported for coolant at inlet temperature
- Flow split can be obtained as ratio of subchannel and average velocities
- Average values reported for assemblies without rod bundle specification

                            |-------  --------  Velocity  (m/s) --  --------  -------|    Bundle  Friction  Eddy Df.
Asm.      Name       Pos.       Avg.       Int.      Edge    Corner    Bypass     Swirl        RE    Factor   (m^2/s)
---------------------------------------------------------------------------------------------------------------------
   1      fuel    ( 1, 1)      7.900      7.655     8.868     7.407       ---     1.115     57046  2.06E-02   0.00062
   2      fuel    ( 2, 1)      7.182      6.959     8.062     6.733       ---     1.014     51860  2.10E-02   0.00056
   3      fuel    ( 2, 2)      7.182      6.959     8.062     6.733       ---     1.014     51860  2.10E-02   0.00056
   4      fuel    ( 2, 3)      7.182      6.959     8.062     6.733       ---     1.014     51860  2.10E-02   0.00056
   5      fuel    ( 2, 4)      7.182      6.959     8.062     6.733       ---     1.014     51860  2.10E-02   0.00056
   6      fuel    ( 2, 5)      7.182      6.959     8.062     6.733       ---     1.014     51860  2.10E-02   0.00056
   7      fuel    ( 2, 6)      7.182      6.959     8.062     6.733       ---     1.014     51860  2.10E-02   0.00056
   8      fuel    ( 3, 1)      4.596      4.454     5.160     4.309       ---     0.649     33190  2.27E-02   0.00036
```

Figure 27. Example subchannel flow characteristics table from summary output

5.1.4 *Table 4: Pressure drop*

Axial pressure drop results for each assembly are reported in Table 4. Results are presented for each axial region (e.g., the pin bundle and each low-fidelity model region) and summed to report the total value. An example is shown in Figure 28 in which there is only one region (the pin bundle).

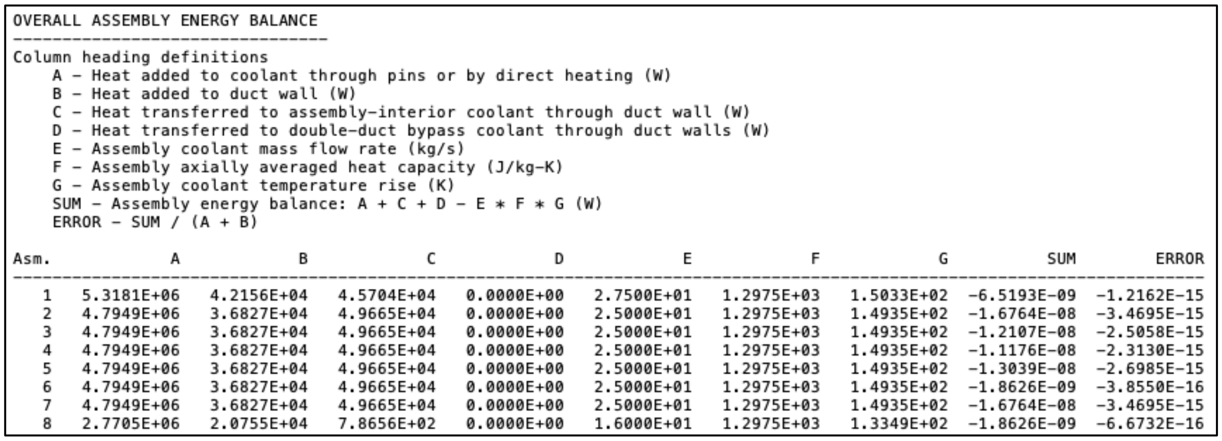


```
PRESSURE DROP (MPa) ACROSS ASSEMBLIES
--------------------------------------
Notes
- "Total" is the total pressure drop accumulated across the assembly
- "Region 1...N" is the pressure drop in each user-specified axial
    region, starting from the bottom

Asm.       Name       Loc.       Total   Region 1
--------------------------------------------------
   1       fuel     ( 1, 1)    0.82304    0.82304
   2       fuel     ( 2, 1)    0.69197    0.69197
   3       fuel     ( 2, 2)    0.69197    0.69197
   4       fuel     ( 2, 3)    0.69197    0.69197
   5       fuel     ( 2, 4)    0.69197    0.69197
   6       fuel     ( 2, 5)    0.69197    0.69197
   7       fuel     ( 2, 6)    0.69197    0.69197
   8       fuel     ( 3, 1)    0.30714    0.30714
```

Figure 28. Example pressure drop table from summary output

5.1.5 Table 5: Assembly and core energy balance

Table 5 presents the energy balance results on the coolant in each assembly. This table is populated if the "calc_energy_balance" option in the Setup block is enabled. The terms in the energy balance include heat added directly to the coolant or via pins (A), heat added to the duct wall (B), heat transferred through ducts to the interior coolant (C), or bypass coolant between ducts in double-ducted assemblies (D), and the energy rise of the coolant (product of E, F, and G). Energy added to the duct is shown but is not included in the balance because it reaches the coolant through heat transfer with the duct. The table sums the results to report the energy balance difference and an error relative to the total assembly power. An example is shown in Figure 29.

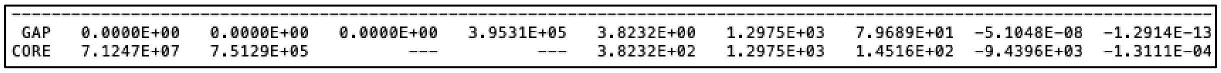


```
OVERALL ASSEMBLY ENERGY BALANCE
--------------------------------
Column heading definitions
    A - Heat added to coolant through pins or by direct heating (W)
    B - Heat added to duct wall (W)
    C - Heat transferred to assembly-interior coolant through duct wall (W)
    D - Heat transferred to double-duct bypass coolant through duct walls (W)
    E - Assembly coolant mass flow rate (kg/s)
    F - Assembly axially averaged heat capacity (J/kg-K)
    G - Assembly coolant temperature rise (K)
    SUM - Assembly energy balance: A + C + D - E * F * G (W)
    ERROR - SUM / (A + B)

Asm.             A            B            C            D            E            F            G          SUM        ERROR
--------------------------------------------------------------------------------------------------------------------------
   1    5.3181E+06   4.2156E+04   4.5704E+04   0.0000E+00   2.7500E+01   1.2975E+03   1.5033E+02  -6.5193E-09  -1.2162E-15
   2    4.7949E+06   3.6827E+04   4.9665E+04   0.0000E+00   2.5000E+01   1.2975E+03   1.4935E+02  -1.6764E-08  -3.4695E-15
   3    4.7949E+06   3.6827E+04   4.9665E+04   0.0000E+00   2.5000E+01   1.2975E+03   1.4935E+02  -1.2107E-08  -2.5058E-15
   4    4.7949E+06   3.6827E+04   4.9665E+04   0.0000E+00   2.5000E+01   1.2975E+03   1.4935E+02  -1.1176E-08  -2.3130E-15
   5    4.7949E+06   3.6827E+04   4.9665E+04   0.0000E+00   2.5000E+01   1.2975E+03   1.4935E+02  -1.3039E-08  -2.6985E-15
   6    4.7949E+06   3.6827E+04   4.9665E+04   0.0000E+00   2.5000E+01   1.2975E+03   1.4935E+02  -1.8626E-09  -3.8550E-16
   7    4.7949E+06   3.6827E+04   4.9665E+04   0.0000E+00   2.5000E+01   1.2975E+03   1.4935E+02  -1.6764E-08  -3.4695E-15
   8    2.7705E+06   2.0755E+04   7.8656E+02   0.0000E+00   1.6000E+01   1.2975E+03   1.3349E+02  -1.8626E-09  -6.6732E-16
```

Figure 29. Example assembly energy balance table from summary output

At the bottom of Table 5 are the energy balance values for the inter-assembly gap and a total result for the entire core as shown in Figure 30. The inter-assembly gap receives no direct heating but can gain energy from the assemblies. The gap energy balance reflects the difference between how much heat is gained by convection and how much heat results in coolant temperature rise. The gap energy balance is not exact if non-flowing inter-assembly gap boundary conditions are used.

The core total energy balance reflects the total heat addition to the core and the total energy added to the coolant in each assembly and in the inter-assembly gap. The energy added to the coolant is calculated for each assembly based on an average value for heat capacity. If variable material properties are used, this energy balance will reflect a small error because the average does not capture the axial variation in heat capacity due to temperature. If power is added to the duct wall, this also can cause error in the core energy balance to because the transfer of that heat to the coolant is lagged with the duct wall calculation. This is the cause of the non-zero core-total energy balance shown in Figure 30.

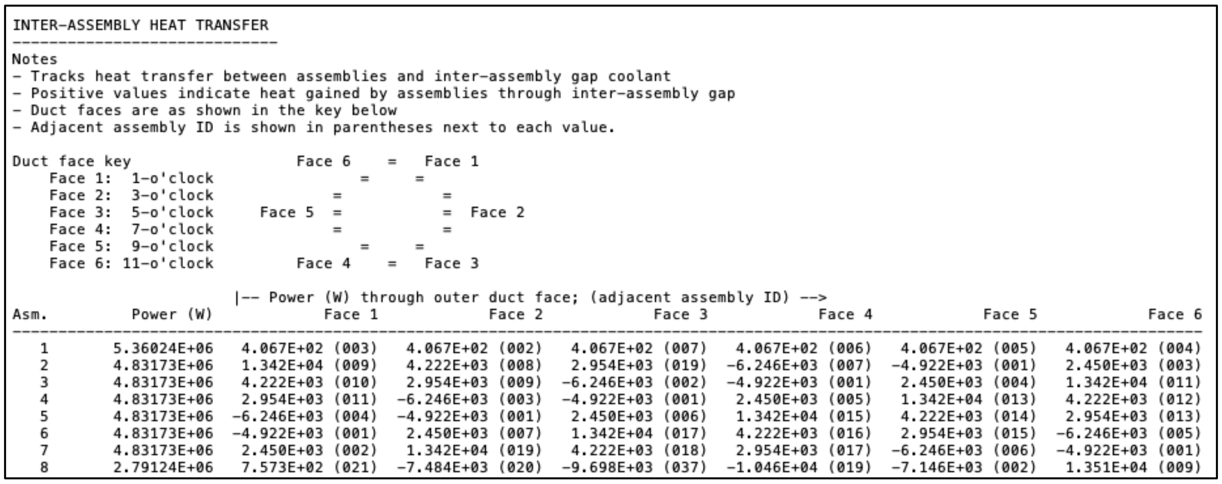


```
-----------------------------------------------------------------------------------------------------------------------
 GAP   0.0000E+00   0.0000E+00   0.0000E+00   3.9531E+05   3.8232E+00   1.2975E+03   7.9689E+01  -5.1048E-08  -1.2914E-13
CORE   7.1247E+07   7.5129E+05          ---          ---   3.8232E+02   1.2975E+03   1.4516E+02  -9.4396E+03  -1.3111E-04
```

Figure 30. Example gap and core energy balance table from summary output

5.1.6 Table 6: Inter-assembly heat transfer

Table 6 shows heat transfer between assemblies and the inter-assembly gap. The values in the table are calculated at each step for the connections between each coolant subchannel and each adjacent duct wall over the axial sweep, then accumulated per hex face. Because the inter-assembly gap corner subchannels are not necessarily symmetrical, heat is distributed between sides according to the interface area of the corner on each hex face. Heat not transferred between assemblies is removed by the inter-assembly gap coolant. The table includes the assembly power to highlight the relative impact of inter-assembly heat transfer. Table 6 is populated if the "calc_energy_balance" option in the Setup block is enabled. An example is shown in Figure 31.

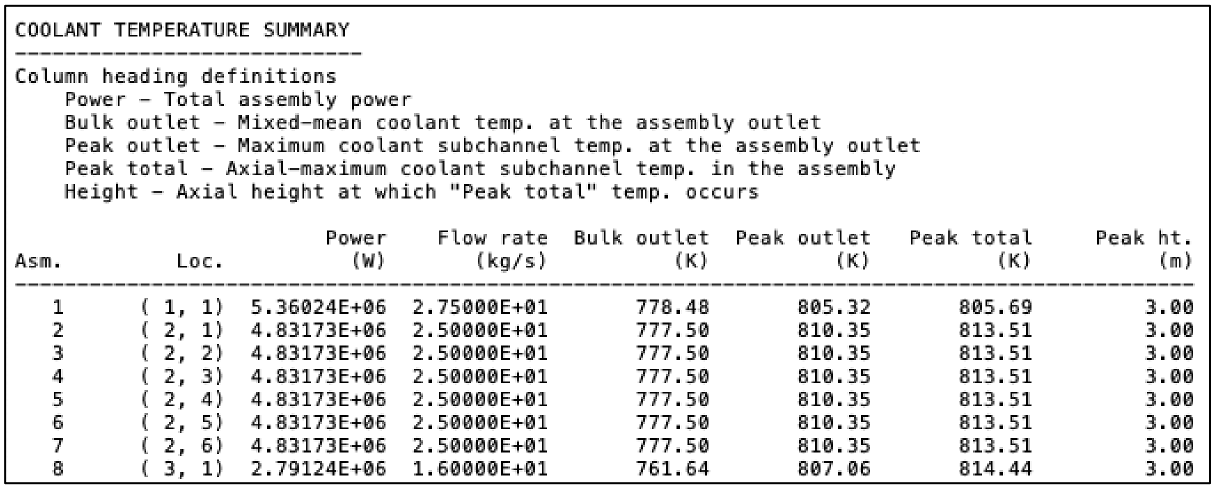


```
INTER-ASSEMBLY HEAT TRANSFER
-----------------------------
Notes
- Tracks heat transfer between assemblies and inter-assembly gap coolant
- Positive values indicate heat gained by assemblies through inter-assembly gap
- Duct faces are as shown in the key below
- Adjacent assembly ID is shown in parentheses next to each value.

Duct face key                Face 6    =   Face 1
    Face 1:  1-o'clock                =     =
    Face 2:  3-o'clock             =           =
    Face 3:  5-o'clock       Face 5  =           =  Face 2
    Face 4:  7-o'clock             =           =
    Face 5:  9-o'clock                =     =
    Face 6: 11-o'clock           Face 4    =   Face 3

                          |-- Power (W) through outer duct face; (adjacent assembly ID) -->
Asm.          Power (W)            Face 1              Face 2              Face 3              Face 4              Face 5              Face 6
---------------------------------------------------------------------------------------------------------------------------------------------
   1        5.36024E+06   4.067E+02 (003)   4.067E+02 (002)   4.067E+02 (007)   4.067E+02 (006)   4.067E+02 (005)   4.067E+02 (004)
   2        4.83173E+06   1.342E+04 (009)   4.222E+03 (008)   2.954E+03 (019)  -6.246E+03 (007)  -4.922E+03 (001)   2.450E+03 (003)
   3        4.83173E+06   4.222E+03 (010)   2.954E+03 (009)  -6.246E+03 (002)  -4.922E+03 (001)   2.450E+03 (004)   1.342E+04 (011)
   4        4.83173E+06   2.954E+03 (011)  -6.246E+03 (003)  -4.922E+03 (001)   2.450E+03 (005)   1.342E+04 (013)   4.222E+03 (012)
   5        4.83173E+06  -6.246E+03 (004)  -4.922E+03 (001)   2.450E+03 (006)   1.342E+04 (015)   4.222E+03 (014)   2.954E+03 (013)
   6        4.83173E+06  -4.922E+03 (001)   2.450E+03 (007)   1.342E+04 (017)   4.222E+03 (016)   2.954E+03 (015)  -6.246E+03 (005)
   7        4.83173E+06   2.450E+03 (002)   1.342E+04 (019)   4.222E+03 (018)   2.954E+03 (017)  -6.246E+03 (006)  -4.922E+03 (001)
   8        2.79124E+06   7.573E+02 (021)  -7.484E+03 (020)  -9.698E+03 (037)  -1.046E+04 (019)  -7.146E+03 (002)   1.351E+04 (009)
```

Figure 31. Example inter-assembly heat transfer table from summary output

5.1.7 Table 7: Coolant temperature summary

Coolant temperature results are summarized for each assembly in Table 7, for which an example is shown in Figure 32. For each assembly, the table reports the average outlet temperature, the peak outlet temperature (the maximum temperature of any subchannel at the assembly outlet), and the peak overall temperature (the maximum temperature of any subchannel at any height in the assembly). The height at which the peak overall temperature occurs is reported as well. The assembly power and flow rate are included again this table for users to be able to make quick assessments of how much power was translated into coolant temperature rise.

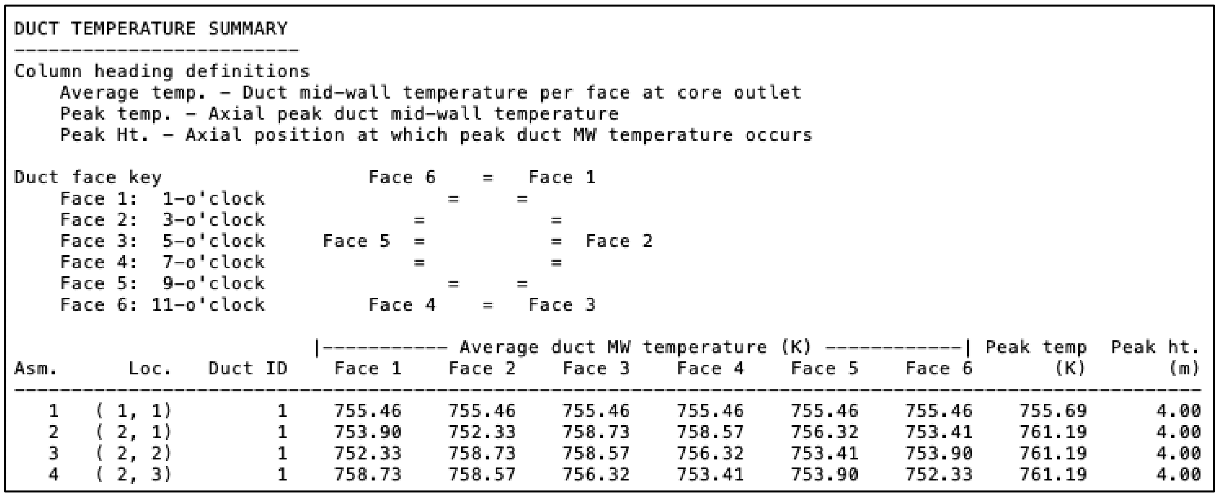


```
COOLANT TEMPERATURE SUMMARY
----------------------------
Column heading definitions
    Power - Total assembly power
    Bulk outlet - Mixed-mean coolant temp. at the assembly outlet
    Peak outlet - Maximum coolant subchannel temp. at the assembly outlet
    Peak total - Axial-maximum coolant subchannel temp. in the assembly
    Height - Axial height at which "Peak total" temp. occurs

                          Power    Flow rate  Bulk outlet  Peak outlet   Peak total    Peak ht.
Asm.          Loc.          (W)       (kg/s)          (K)          (K)          (K)         (m)
-----------------------------------------------------------------------------------------------
   1       ( 1, 1)  5.36024E+06  2.75000E+01       778.48       805.32       805.69        3.00
   2       ( 2, 1)  4.83173E+06  2.50000E+01       777.50       810.35       813.51        3.00
   3       ( 2, 2)  4.83173E+06  2.50000E+01       777.50       810.35       813.51        3.00
   4       ( 2, 3)  4.83173E+06  2.50000E+01       777.50       810.35       813.51        3.00
   5       ( 2, 4)  4.83173E+06  2.50000E+01       777.50       810.35       813.51        3.00
   6       ( 2, 5)  4.83173E+06  2.50000E+01       777.50       810.35       813.51        3.00
   7       ( 2, 6)  4.83173E+06  2.50000E+01       777.50       810.35       813.51        3.00
   8       ( 3, 1)  2.79124E+06  1.60000E+01       761.64       807.06       814.44        3.00
```

Figure 32. Example coolant temperature table from summary output file

5.1.8 Table 8: Duct temperature summary

The duct temperature summary table reports, for each duct wall in each assembly, the average duct mid-wall temperatures on each hex face at the assembly outlet, as well as the overall peak temperature experienced by any duct element at any point in the assembly and the height at which that peak temperature occurred. An example of the duct temperature summary table is shown in Figure 33. in Figure 33, "duct ID" is the index of the duct; for double-ducted assemblies, two entries will be present: one for the inner duct (1) and one for the outer duct (2).

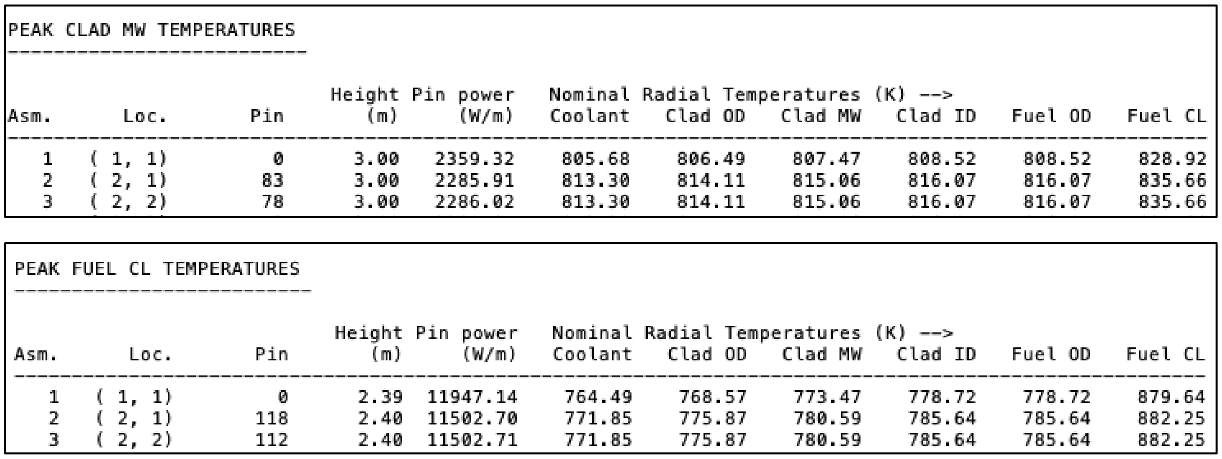


```
DUCT TEMPERATURE SUMMARY
--------------------------
Column heading definitions
    Average temp. - Duct mid-wall temperature per face at core outlet
    Peak temp. - Axial peak duct mid-wall temperature
    Peak Ht. - Axial position at which peak duct MW temperature occurs

Duct face key                  Face 6    =   Face 1
    Face 1:  1-o'clock                =     =
    Face 2:  3-o'clock             =           =
    Face 3:  5-o'clock      Face 5 =           =  Face 2
    Face 4:  7-o'clock             =           =
    Face 5:  9-o'clock                =     =
    Face 6: 11-o'clock         Face 4    =   Face 3

                              |----------- Average duct MW temperature (K) ------------| Peak temp  Peak ht.
Asm.      Loc.    Duct ID    Face 1    Face 2    Face 3    Face 4    Face 5    Face 6       (K)        (m)
----------------------------------------------------------------------------------------------------------
   1    ( 1, 1)         1    755.46    755.46    755.46    755.46    755.46    755.46    755.69       4.00
   2    ( 2, 1)         1    753.90    752.33    758.73    758.57    756.32    753.41    761.19       4.00
   3    ( 2, 2)         1    752.33    758.73    758.57    756.32    753.41    753.90    761.19       4.00
   4    ( 2, 3)         1    758.73    758.57    756.32    753.41    753.90    752.33    761.19       4.00
```

Figure 33. Example duct temperature table from summary output file

5.1.9 Table 9: Radial pin temperatures

For assemblies for which pin model parameters are specified, the results of the pin temperature calculation are summarized in two tables. The first table prints the radial pin temperatures (from adjacent coolant to fuel centerline) for each assembly in the pin and at the height of the peak clad mid-wall temperature. The second table reports radial pin temperatures for each assembly in the pin and at the height of the peak fuel centerline temperature. Examples of these two tables are shown in Figure 34.



```
PEAK CLAD MW TEMPERATURES
--------------------------

                                Height  Pin power   Nominal  Radial Temperatures (K) -->
Asm.       Loc.         Pin       (m)      (W/m)    Coolant   Clad OD   Clad MW   Clad ID   Fuel OD   Fuel CL
-------------------------------------------------------------------------------------------------------------
   1     ( 1, 1)          0      3.00    2359.32     805.68    806.49    807.47    808.52    808.52    828.92
   2     ( 2, 1)         83      3.00    2285.91     813.30    814.11    815.06    816.07    816.07    835.66
   3     ( 2, 2)         78      3.00    2286.02     813.30    814.11    815.06    816.07    816.07    835.66
```

```
PEAK FUEL CL TEMPERATURES
--------------------------

                                Height  Pin power   Nominal  Radial Temperatures (K) -->
Asm.       Loc.         Pin       (m)      (W/m)    Coolant   Clad OD   Clad MW   Clad ID   Fuel OD   Fuel CL
-------------------------------------------------------------------------------------------------------------
   1     ( 1, 1)          0      2.39   11947.14     764.49    768.57    773.47    778.72    778.72    879.64
   2     ( 2, 1)        118      2.40   11502.70     771.85    775.87    780.59    785.64    785.64    882.25
   3     ( 2, 2)        112      2.40   11502.71     771.85    775.87    780.59    785.64    785.64    882.25
```

Figure 34. Example pin temperature tables in the summary output.

5.2 Dumped temperature data: CSV files

If the user requires the entire temperature profile of the core, they should use the "dump" inputs in the Options input block to produce CSV files of temperature data. Because these temperatures are organized by subchannel, they are best processed via DASSH, which already has the capability to map them. The primary use for these files is the built-in visualization capability included in DASSH.

For users who need to perform their own data analysis, the format of these files is described in Table 20. The first few columns of each row are dedicated to data that characterize the temperatures included in the remainder of the row. For nearly all data files, the first two columns are: (1) assembly ID; and (2) axial position (m).

Contact the developers as needed for assistance. Note that in the future, DASSH will likely be revised to dump data to HDF5 files rather than CSVs. This section will be updated when this capability is implemented.

Table 20. Description of CSV data files produced by DASSH

File name	Descriptive columns	File description
temp_coolant_int.csv	1. Assembly ID 2. Axial height (m) 3. Axial region ID	Temperatures of coolant subchannels within the innermost assembly duct wall according to the map shown in Figure 1.
temp_duct_mw.csv	1. Assembly ID 2. Axial height (m) 3. Axial region ID 4. Duct ID	Duct mid-wall temperatures for each duct mesh in each assembly, according to the arrangement of duct meshes shown in Figure 1.
temp_coolant_byp.csv	1. Assembly ID 2. Axial height (m) 3. Axial region ID 4. Bypass ID	Temperatures of coolant subchannels that comprise the bypass gap between double-ducted assembly duct walls; the arrangement is analogous to that of the duct meshes shown in Figure 1.
temp_coolant_gap.csv	1. Assembly ID 2. Axial height (m)	Temperature of inter-assembly gap coolant subchannels adjacent to each assembly. Row lengths vary; empty positions filled with zeros.

temp_coolant_gap_fine.csv	1. Axial height	Temperature of inter-assembly gap coolant subchannels on the inter-assembly gap coolant mesh
temp_pin.csv	1. Assembly ID 2. Axial height (m) 3. Pin ID	Radial temperatures in each pin; data columns are: 4. Avg. adjacent coolant temperature 5. Clad OD temperature 6. Clad MW temperature 7. Clad ID temperature 8. Fuel OD temperature 9. Fuel CL temperature
temp_average.csv	1. Assembly ID 2. Axial height (m) 3. Axial region ID	Assembly-average temperatures data columns are average temperatures of: 4. Interior coolant 5. All coolant, incl. bypass channels 6. Innermost duct mid-wall 7. Outermost duct mid-wall 8. Clad mid-wall (of all pins) 9. Fuel centerline (of all pins)
temp_maximum.csv	1. Assembly ID 2. Axial height (m) 3. Axial region ID	Assembly-maximum temperatures; data columns are maximum temperatures of: 4. Interior coolant subchannel 5. Innermost duct mid-wall 6. Clad mid-wall (of all pins) 7. Fuel centerline (of all pins)

5.3 dassh_reactor.pkl binary file

If DASSH is run from the command line with the "--save_reactor" input flag, or if plots are requested in the input file, a file called "dassh_reactor.pkl" will be saved in the working directory. This is a binary file containing the main Python code object (the "Reactor" object) used in the calculation. It stores all information about the assembly geometry, correlated parameters, and temperatures evaluated at the last axial plane in the problem. DASSH uses these stored details along with the data dumped to CSVs to produce figures. This object may also be interacted with directly in the Python interpreter by invoking the Python commands below.

```
>>> import dassh
>>> r = dassh.reactor.load()
```

Most users will not need to interact with the Reactor object through the Python interpreter. The primary reason to do so is to gain additional insights into the system state at the end of the calculation for debugging purposes. Users interested in interacting with the Reactor object – and the objects it contains – should consult the source code, which includes information about the main attributes.

6 Examples

To aid users in preparing their own problems, two simple examples are provided here for reference. Input and data files for these problems are available in the DASSH GitHub repository: https://github.com/dassh-dev/examples. This section will be updated as capabilities and use cases are added.

6.1 Single-assembly problem

The problem shown is for a single assembly producing 6 MWth with a flat power distribution and using adiabatic radial boundary conditions and an axial boundary condition of 150 K temperature rise across the core. The geometry is modeled after the FASTER fuel assembly design [44]. Fuel temperatures are calculated assuming no fuel-clad gap.

```
[Setup]
    calc_energy_balance = True
    log_progress        = 200
    [[Dump]]
        coolant = True
        pins    = True


[Power]
    total_power = 6.0e6
    [[ARC]]
        fuel_material   = metal
        fuel_alloy      = zr
        coolant_heating = sodium
        power_model     = pin_only
        pmatrx = cccc/PMATRX
        geodst = cccc/GEODST
        ndxsrf = cccc/NDXSRF
        znatdn = cccc/ZNATDN
        labels = cccc/LABELS
        nhflux = cccc/NHFLX0N
        ghflux = cccc/NHFLX0G


[Core]
    coolant_inlet_temp = 628.15
    coolant_material   = sodium
    length             = 4.0000
    gap_model          = none
    assembly_pitch     = 0.1200


# continued below
# continued from above

[Assembly]
    [[fuel]]
        num_rings       = 10
        pin_pitch       = 0.00650
        pin_diameter    = 0.00540
        clad_thickness  = 0.00035
        wire_pitch      = 0.20320
        wire_diameter   = 0.00100
        duct_ftf        = 0.10960, 0.11560
        duct_material   = ht9
        corr_mixing     = MIT
        corr_friction   = NOV
        corr_flowsplit  = MIT
        corr_nusselt    = DB
        htc_params_duct = 0.025, 0.8, 0.8, 7.0
        wire_direction  = counterclockwise
        shape_factor    = 1.0
        [[[FuelModel]]]
            fcgap_thickness = 0.0
            clad_material   = ht9
            r_frac   =  0.0, 0.33333, 0.66667
            pu_frac  = 0.20,    0.20,    0.20
            zr_frac  = 0.10,    0.10,    0.10
            porosity = 0.25,    0.25,    0.25


[Assignment]
    [[ByPosition]]
        fuel = 1, 1, 1, DELTA_TEMP=150.0


[Plot]
    [[subchannel_temps]]
        type        = SubchannelPlot
        assembly_id = 1
        z           = 1.50, 3.50       # m
        cmap        = jet              # default
        units       = Celsius          # include because different than default (K)
        cbar_lbnd   = 355.0            # celsius, as indicated
        cbar_ubnd   = 545.0            # celsius, as indicated
        cbar_label  = Coolant temperature (˚C)
        pins        = True             # Overlay pins on subchannels
        pin_alpha   = 0.8              # Opacity of pin fill

     [[fuel_temps]]
        type        = PinPlot
        value       = clad_mw
        assembly_id = 1
        z           = 1.50, 2.50       # m
        cmap        = rainbow          # cmap option provided by matplotlib
        units       = Celsius          # include because different from default (K)
        cbar_label  = Clad MW temperature (˚C)
```

Figure 35. Single-assembly model input

Running the above input file results in the generation of the DASSH summary output file “dassh.out”, a Python binary file that contains the main DASSH code object, called “dassh_reactor.pkl”, many CSVs storing temperature data, and 4 figures: 2 subchannel temperature figures and 2 figures showing clad temperatures.

6.2 37-assembly problem

The problem shown is for a 37-assembly core comprised of 19 fuel assemblies surrounded by one row of reflector assemblies. Power distributions are supplied via ARC binary files. This problem demonstrates user-specification of material properties, the use of the flowing inter-assembly gap model, and the syntax to request different types of core-wide figures.

```
[Setup]
    log_progress          = 50     # Print update every 50 steps
    calc_energy_balance   = True   # Calculate energy balance
    conv_approx           = True   # Apply convection approximation...
    conv_approx_dz_cutoff = 0.001  # ...when step size < 0.1 cm.
    [[Dump]]
        coolant  = True
        average  = True
        maximum  = True
        pins     = True
        duct     = True
        interval = 10.0


[Materials]
    [[sodium_x]]
        thermal_conductivity = 75.35648125
        viscosity            = 0.00033060424456306015
        heat_capacity        = 1297.485560623364
        density              = 874.0794169687499
    [[ht9_x]]
        thermal_conductivity = 26.0665655


[Power]
    total_power = 72.0e6
    [[ARC]]
        fuel_material   = metal
        fuel_alloy      = zr
        coolant_heating = sodium
        pmatrx = cccc/PMATRX
        geodst = cccc/GEODST
        ndxsrf = cccc/NDXSRF
        znatdn = cccc/ZNATDN
        labels = cccc/LABELS
        nhflux = cccc/NHFLX0N
        ghflux = cccc/NHFLX0G


[Core]
    coolant_inlet_temp = 628.15
    coolant_material   = sodium_x
    length             = 4.0000
    gap_model          = flow
    bypass_fraction    = 0.0100
    assembly_pitch     = 0.1200


# continued below
# continued from above

[Assembly]

    [[fuel]]
        num_rings       = 10
        pin_pitch       = 0.00650
        pin_diameter    = 0.00540
        clad_thickness  = 0.00035
        wire_pitch      = 0.20320
        wire_diameter   = 0.00100
        duct_ftf        = 0.10960, 0.11560
        duct_material   = ht9_x
        htc_params_duct = 0.025, 0.8, 0.8, 7.0
        shape_factor = 1.1
        [[[FuelModel]]]
            fcgap_thickness = 0.0
            clad_material   = ht9_x
            r_frac   =  0.0, 0.33333, 0.66667
            pu_frac  = 0.20,    0.20,    0.20
            zr_frac  = 0.10,    0.10,    0.10
            porosity = 0.25,    0.25,    0.25

    [[reflector]]
        num_rings       = 6
        pin_pitch       = 0.011200
        pin_diameter    = 0.010950
        clad_thickness  = 0.001006
        wire_pitch      = 0.204190
        wire_diameter   = 0.000123
        duct_ftf        = 0.109600, 0.115600
        duct_material   = ht9_x
        htc_params_duct = 0.025, 0.8, 0.8, 7.0


[Assignment]
    [[ByPosition]]
        fuel      = 1,  1,  1, FLOWRATE = 27.5
        fuel      = 2,  1,  6, FLOWRATE = 25.0
        fuel      = 3,  1, 12, FLOWRATE = 16.0
        reflector = 4,  1, 18, FLOWRATE = 0.50


[Plot]
     [[subchannel_temps]]
        type       = CoreSubchannelPlot
        z          = 2.000, 3.500  # m
        cmap       = jet
        units      = celsius
        cbar_lbnd  = 355.0
        cbar_ubnd  = 545.0
        cbar_label = Coolant temperature (˚C)

     [[fuel_temps]]
        type       = CorePinPlot
        value      = fuel_cl
        z          = 1.500, 2.500  # m
        cbar_label = Fuel CL temperature (˚C)
        units      = celsius

     [[asm_power]]
         type        = CoreHexPlot
         value       = total_power
         cbar_label  = Assembly Power

     [[avg_outlet_coolant_temp]]
         type        = CoreHexPlot
         value       = avg_coolant_temp
         z           = 4.000  # m
         cbar_label  = Avg. coolant temperature (˚C)
         units       = Celsius
```

Figure 36. 37-assembly model input

Acknowledgements

The authors would like to acknowledge the careful review of this document performed by Nicolas Martin. Additionally, the early feedback on DASSH from Michael Jarrett was incredibly helpful. This document has been created by UChicago Argonne, LLC, Operator of Argonne National Laboratory ("Argonne"). Argonne National Laboratory's work was supported by the U.S. Department of Energy, Office of Nuclear Energy under contract DE-AC02-06CH11357. The work reported in this document is the result of ongoing efforts supporting the Versatile Test Reactor.

References

[1] M. Atz, M. A. Smith and F. Heidet, "DASSH software for ducted assembly thermal hydraulics calculations – overview and benchmark," *Transactions of the American Nuclear Society,* vol. 123, pp. 1673-1676, 2020.

[2] M. Atz, M. A. Smith and F. Heidet, "Ducted Assembly Steady-State Heat Transfer Software (DASSH) - Theory Manual," ANL/NSE-21/33, Argonne National Laboratory, 2021.

[3] E. U. Khan, W. M. Rohsenow, A. A. Sonin and N. E. Todreas, "A porous body model for predicting temperature distribution in wire-wrapped fuel rod assemblies," *Nuclear Engineering and Design,* vol. 35, pp. 1-12, 1975.

[4] G. Palmiotti, E. Lewis and C. Carrico, "VARIANT: VARIational Anisotropic Nodal Transport for Multidimensional Cartesian and Hexagonal Geometry Calculations," ANL-95/40, Argonne National Laboratory, 1995.

[5] Argonne National Laboratory, "Reactor Physics and Fuel Cycle Analysis Software," 2020. [Online]. Available: http://www.ne.anl.gov/capabilities/rpfca/codes/index.html. [Accessed 2020 17 August].

[6] W. S. Yang, *Fortran 77 Version of SE2-ANL,* Argonne National Laboratory, 1993.

[7] K. L. Basehore and N. E. Todreas, "SUPERENERGY-2: A multiassembly, steady-state computer code for LMFBR core thermal-hydraulic analysis," 1980.

[8] B. V. Richard, "Reactor Hot Spot Analysis," FRA-TM-152, Argonne National Laboratory, 1985.

[9] K. L. Derstine, "DIF3D - A Code to Solve One-, Two-, and Three-Dimensional Finite-Difference Diffusion Theory Problems," ANL-82-64, Argonne National Laboratory, 1984.

[10] Continuum Analytics, "Anaconda," 2017. [Online]. Available: https://www.anaconda.com. [Accessed 17 August 2020].

[11] Continuum Analytics, "Miniconda," 2017. [Online]. Available: https://docs.conda.io/en/latest/miniconda.html. [Accessed 17 August 2020].

[12] "pip," 2020. [Online]. Available: https://pip.pypa.io/en/stable/. [Accessed 17 August 2020].

[13] T. E. Oliphant, A Guide to NumPy, Trelgol Publishing, 2006.

[14] M. Foord, N. Larosa, R. Dennis and E. Courtwright, "ConfigObj," 2014. [Online]. Available: https://configobj.readthedocs.io/en/latest/. [Accessed 17 August 2020].

[15] H. Krekel, "Pytest," 2020. [Online]. Available: https://docs.pytest.org/en/stable/contents.html. [Accessed 17 August 2020].

[16] W. McKinney, "Data Structures for Statistical Computing in Python," *Proceedings of the 9th Python in Science Conference,* pp. 56-61, 2010.

[17] J. D. Hunter, "Matplotlib: A 2D graphics environment," *Computing in Science & Engineering,* vol. 9, no. 3, pp. 90-95, 2007.

[18] Continuum Analytics, "Managing environments," 2017. [Online]. Available: https://docs.conda.io/projects/conda/en/latest/user-guide/tasks/manage-environments.html. [Accessed 21 August 2020].

[19] M. A. Smith, C. Lee and R. N. Hill, "GAMSOR: Gamma Source Preparation and DIF3D Flux Solution," ANL/NE-16/50, Argonne National Laboratory, 2016.

[20] J. K. Fink and L. Leibowitz, "Thermodynamic and Transport Properties of Sodium Liquid and Vapor," ANL-RE-95-2, Argonne National Laboratory, 1995.

[21] O. J. Foust, Ed., Sodium-NaK Engineering Handbook Volume I - Sodium Chemistry and Physical Properties, Gordon and Breach, Science Publishers, Inc., 1972.

[22] V. Sobolev, "Database of thermophysical properties of liquid metal coolants for GEN-IV," SCK-CEN-BLG-1069, 2010.

[23] L. Leibowitz and R. A. Blomquist, "Thermal conductivity and thermal expansion of stainless steels D9 and HT9," *International Journal of Thermophysics,* vol. 9, no. 5, pp. 873-883, 1988.

[24] G. L. Hofman, M. C. Billone, J. F. Koenig, J. M. Kramer, J. D. B. Lambert, L. Leibowitz, Y. Orechwa, D. R. Pedersen, D. L. Porter, H. Tsai and A. E. Wright, "Metallic Fuels Handbook," ANL-NSE-3, Argonne National Laboratory, 2019.

[25] K. C. Mills, Recommended Values of Thermophysical Properties for Selected Commercial Alloys, Woodhead Publishing, 2002.

[26] C. S. Kim, "Thermophysical Properties of Stainless Steels," ANL-75-55, Argonne National Laboratory, 1975.

[27] M.-. H. Chun and K.-. W. Seo, "An experimental study and assessment of existing friction factor correlations for wire-wrapped fuel assemblies," *Annals of Nuclear Energy,* vol. 28, pp. 1683-1695, 2001.

[28] S. K. Chen, N. E. Todreas and N. T. Nguyen, "Evaluation of existing correlations for the prediction of pressure drop in wire-wrapped hexagonal array pin bundles," *Nuclear Engineering and Design,* vol. 267, pp. 109-131, 2014.

[29] E. H. Novendstern, "Turbulent flow pressure drop model for fuel rod assemblies utilizing a helical wire-wrap spacer system," *Nuclear Engineering and Design,* vol. 22, pp. 19-27, 1972.

[30] K. Rehme, "Pressure drop correlations for fuel element spacers," *Nuclear Technology,* vol. 17, no. 1, pp. 15-23, 1973.

[31] F. C. Engel, R. A. Markley and A. A. Bishop, "Laminar, transition, and turbulent parallel flow pressure drop across wire-wrap-spaced rod bundles," *Nuclear Science and Engineering,* vol. 69, no. 2, pp. 290-296, 1979.

[32] S.-K. Cheng and N. E. Todreas, "Hydrodynamic models and correlations for bare and wire-wrapped hexagonal rod bundles - bundle friction factors, subchannel friction factors and mixing parameters," *Nuclear Engineering and Design,* vol. 92, pp. 227-251, 1986.

[33] S. K. Chen, Y. M. Chen and N. E. Todreas, "The upgraded Cheng and Todreas correlation for pressure drop in hexagonal wire-wrapped rod bundles," *Nuclear Engineering and Design,* vol. 335, pp. 356-373, 2018.

[34] S.-K. Cheng, Constitutive correlations for wire-wrapped subchannel analysis under forced and mixed convection conditions, Massachusetts Institute of Technology, 1984.

[35] C. Chiu, T. E. Neil and W. M. Rohsenow, "Turbulent flow split model and supporting experiments for wire-wrapped core assemblies," *Nuclear Technology,* vol. 50, no. 1, pp. 40-52, 1980.

[36] R. Lyon, "Forced convection heat transfer theory and experiments with liquid metals," ORNL-361, Oak Ridge National Laboratory, 1949.

[37] R. Lyon, "Liquid metal heat-transfer coefficients," *Chemical Engineering Progress,* vol. 47, no. 2, pp. 75-79, 1951.

[38] R. C. Martinelli, "Heat transfer to molten metals," *Transactions of the American Society of Mechanical Engineers,* vol. 69, p. 947–959, 1947.

[39] J. Pacio, L. Marocco and T. Wetzel, " Review of data and correlations for turbulent forced convective heat transfer of liquid metals in pipes," *Heat Mass Transfer,* vol. 51, pp. 153-164, 2015.

[40] D. J. Zigrang and N. D. Sylvester, "A Review of Explicit Friction Factor Equations," *Journal of Energy Resources Technology,* vol. 107, no. 2, pp. 280-283, 1985.

[41] The RELAP5 Development Team, RELAP5/MOD3 Code Manual - Code Structure, System Models, and Solution Methods, vol. 1, NUREG/CR-5535, INEL-95/0174, Idaho National Laboratory, 1995.

[42] J. Calahan, T. Fanning, M. Farmer, C. Grandy, E. Jin, T. Kim, R. Kellogg, L. Krajtl, S. Lomperski, A. Moisseytsev, Y. Momozaki, Y. Park, C. Reed, F. Salev, R. Seidensticker, J. Sienicki, Y. Tang, C. Tzanos, T. Wei, W. Yang and Chikazawa, Advanced Burner Reactor 1000MWth Reference Concept, C. Grandy and R. Seidensticker, Eds., ANL-AFCI-202 (ANL-ABR-4) Argonne National Laboratory, 2007.

[43] The Matplotlib development team, "matplotlib: Colormap references," 5 1 2020. [Online]. Available: https://matplotlib.org/3.1.1/gallery/color/colormap_reference.html. [Accessed 17 11 2020].

[44] C. Grandy, H. Belch, A. J. Brunett, F. Heidet, R. Hill, E. Hoffman, E. Jin, W. Mohamed, A. Moisseytsev, S. Passerini, J. Sienicki, T. Sumner, R. Vilim and S. Hayes, FASTER Test Reactor Preconceptual Design Report, ANL-ART-86, Argonne National Laboratory, 2016.

This page left blank intentionally