FINITE ELEMENT ANALYSIS

Mastering Abaqus Thermal Subroutines: A Comprehensive Guide to DFLUX, HETVAL, and UMDFLUX for Advanced Heat Transfer Modeling

Published: February 1, 2025 | Index ID: #707

In the realm of advanced Finite Element Analysis (FEA), the ability to accurately simulate thermal phenomena is critical for industries ranging from aerospace and automotive to additive manufacturing and energy. While Abaqus provides a robust suite of built-in tools for thermal analysis, complex real-world scenarios—such as moving laser heat sources, non-uniform surface radiation, or phase-change-induced heat generation—often exceed the capabilities of the standard Graphical User Interface (GUI). This is where **user subroutines** become indispensable. For thermal analysts, the **DFLUX**, **HETVAL**, and **UMDFLUX**subroutines represent the primary tools for implementing custom thermal boundary conditions and internal heat sources.

The Fundamental Role of Thermal Subroutines in Abaqus

Abaqus/Standard and Abaqus/Explicit solve the energy balance equation for a continuum. In many cases, the heat flux entering a surface or the heat generated within a volume depends on variables that are not directly available through the standard load definitions. These variables might include the spatial coordinates of an integration point, the current time in the simulation, or the local temperature of the material. By utilizing Fortran-based subroutines, engineers can define mathematical functions that describe these dependencies with surgical precision.

Understanding which subroutine to use is the first step in successful thermal modeling. **DFLUX** is primarily used for defining non-uniform distributed heat fluxes (surface or body), whereas **HETVAL** is designed specifically for internal heat generation, such as that produced by chemical reactions or phase transformations. For scenarios involving multiple moving heat sources, such as multi-track laser welding, **UMDFLUX**provides a more efficient framework than calling multiple individual DFLUX instances.

Deep Dive into the DFLUX Subroutine

The **DFLUX**subroutine is perhaps the most widely used thermal subroutine in Abaqus. It allows for the definition of heat flux as a function of time, position, and temperature. Its signature in Fortran is critical for any developer to understand:

```
SUBROUTINE DFLUX(FLUX,SOL,KSTEP,KINC,TIME,NOEL,NPT,COORDS,JLTYP,TEMP,PRESS,SNAME)
```

Key Variable Definitions and Their Mechanics

- FLUX(1): This is the magnitude of the flux. The user must define this value within the subroutine. For a surface flux, the units are typically Energy/Time/Area. For a body flux, the units are Energy/Time/Volume.
- FLUX(2): This represents the rate of change of the flux with respect to the temperature (dFlux/dTemp). Providing an accurate derivative significantly improves the convergence rate of the Newton-Raphson solver in Abaqus/Standard.
- SOL: This is the current value of the temperature at the integration point. As noted in technical forums, if the flux depends on the temperature of the previous step or the current iteration, SOL is the variable to use.
- COORDS: An array containing the current coordinates of the integration point. This is essential for modeling moving heat sources, where the flux location is a function of TIME(1).

Implementation of Moving Heat Sources

A common application of DFLUX is the simulation of a moving Gaussian heat source, often used in welding simulations. In this scenario, the heat flux is defined by an equation where the center of the heat source changes over time. By calculating the distance between the integration point COORDS and the moving center (v*t, 0, 0), the developer can apply a high-intensity flux only to the relevant localized area at each time increment.

Comparative Analysis: DFLUX vs. HETVAL vs. UMDFLUX

Choosing the right subroutine requires a clear understanding of their structural differences. The following table provides a comparison of their primary use cases and limitations.

Feature	DFLUX	HETVAL	UMDFLUX
Primary Purpose	Surface or Volumetric Heat Flux	Internal Heat Generation (e.g., Phase Change)	Multiple Moving Heat Sources
Input Variables	Time, Coords, Temp, Step/Inc	Temp, State Variables (STATEV), Time	Global Source Management
Complexity	Moderate - High	Low - Moderate	High
Sensitivity Analysis	Provides Flux(2) for dF/dT	Provides FLUX(2) for dH/dT	Advanced Convergence Control
Common Use Case	Laser Welding, Convective Cooling	Curing of Polymers, Latent Heat	Selective Laser Melting (SLM)

The HETVAL Subroutine and Internal Heat Generation

The **HETVAL** subroutine is specifically tailored for defining internal heat generation. Unlike DFLUX, which is often used for external loads, HETVAL is usually linked to material behavior. It is frequently paired with the **USDFLD** or **UMAT** subroutines to manage state variables (STATEV).

In a HETVAL routine, the user defines FLUX(1) as the heat generation rate per unit volume. A classic example is the exothermic reaction during the curing of a composite material. The heat generated depends on the degree of cure, which is tracked as a state variable. As the degree of cure increases, the heat generation rate changes. HETVAL provides the TEMP and DPRED (increments of predefined fields) to allow for complex, coupled thermal-chemical modeling.

The Challenge of Stress and Strain Inputs

A frequent question in technical communities is whether **HETVAL** or **DFLUX** can use stresses or strains as input variables. By default, the answer is no; the standard arguments for these subroutines do not include the stress or strain tensors. However, in a **fully coupled thermo-mechanical analysis**, this limitation can be bypassed using one of two advanced methods:

1. Utility Routines (GETVRM): In certain versions and contexts, GETVRM can be used to access material point information. However, this is often restricted in standard DFLUX/HETVAL calls.
2. Common Blocks: Developers can use a UVARM or UMAT subroutine to write stress/strain data into a Fortran COMMON block or a Module, which is then read by the thermal subroutine during the same increment. This requires careful management of the solver's integration sequence.

Advanced Case Study: Modeling Self-Shadowing in Open Cross-Sections

As highlighted in research data, modeling heat flux on open cross-sections (like tubes or C-channels) involves the challenge of **self-shadowing**. If a heat source is directional (like solar radiation or a plasma spray), certain parts of the geometry may block the flux from reaching other parts. To implement this in **DFLUX**, the developer must:

- Define the vector of the incoming heat flux.
- Calculate the surface normal at the integration point (this may require passing normals via state variables or calculating them based on element face geometry).
- Implement a visibility algorithm. For complex geometries, this often involves a pre-processing step or a ray-

tracing logic within the subroutine to determine if the COORDS of the current integration point are in the "shadow" of another element.

Simulating Multiple Heat Sources with UMDFLUX

When a process involves dozens or hundreds of simultaneous heat sources—common in multi-laser powder bed fusion—the standard **DFLUX** approach becomes computationally inefficient and difficult to manage via the input file. **UMDFLUX**(User Multiple DFLUX) was introduced to address this. It allows the user to define a collection of heat sources and iterate through them within a single subroutine call. This centralized management allows for better optimization of search algorithms, ensuring that the solver doesn't waste cycles calculating flux values for sources that are too far away to affect a specific integration point.

Technical Workflow for UMDFLUX Implementation

1. Define Source Parameters: Store the intensity, radius, and trajectories of all sources in an external data file or a Fortran Module.
2. Spatial Partitioning: Implement a grid-based or tree-based search within the subroutine to quickly identify which sources are near the current integration point.
3. Summation of Contributions: Calculate the flux contribution from each active source and return the total FLUX value to the Abaqus solver.

Field Guide: Troubleshooting and Debugging Subroutines

Developing subroutines is notoriously prone to errors, from Fortran syntax mistakes to solver divergence. Use the following checklist to ensure a stable simulation:

- Compiler Compatibility: Ensure your version of Intel Fortran is compatible with your Abaqus version and that the environment variables are correctly set (e.g., via abaqus verify -user_std).
- Units Consistency: Abaqus is dimensionless. If your geometry is in millimeters, your flux units in DFLUX must reflect that (e.g., mW/mm² instead of W/m²).
- Convergence and Flux(2): If the simulation fails to converge in the first few increments, check your FLUX(2) definition. If the derivative of flux with respect to temperature is complex, even a numerical approximation is better than leaving it as zero.
- Write Statements: Use WRITE(7,*) to output variable values to the .msg file for debugging. This allows you to track the values of COORDS or TEMP during the simulation to ensure the logic is behaving as expected.

Mathematical Foundation of Heat Flux Calculations

The heat flux calculation inside a subroutine usually follows a Gaussian distribution for concentrated sources:

*q(r) = Q_total * (;r ò ƒÀ * r_0^2)) * exp(-k * (r^2 / r_0^2))*

Where:**q(r):** Local flux at distance r from the center.**;s**Efficiency/Absorption coefficient.**r_0:** Characteristic radius of the heat source.**k:** Concentration factor (typically 2 or 3 depending on the definition of the beam profile).

Inside **DFLUX**, r is calculated as SQRT((COORDS(1)-Xc)**2 + (COORDS(2)-Yc)**2), where (Xc, Yc) are the time-dependent coordinates of the source center.

Summary and Broader Engineering Implications

The mastery of **DFLUX**, **HETVAL**, and **UMDFLUX**subroutines transforms Abaqus from a standard simulation tool into a powerful, customizable engine for thermal research and development. By moving beyond the GUI, engineers

can simulate the intricate physics of modern manufacturing processes and material behaviors with high fidelity. While the learning curve for Fortran integration and the Abaqus solver mechanics is steep, the ability to model moving sources, temperature-dependent reactions, and complex shadowing effects provides a significant competitive advantage in predictive engineering.

As FEA continues to evolve toward multi-physics and multi-scale modeling, the integration of thermal subroutines with mechanical and microstructural subroutines will become the standard. Engineers who can navigate the data exchange between these routines—managing state variables and common blocks effectively—will be at the forefront of the next generation of digital twin development and virtual prototyping.

Tags: #Abaqus Subroutines #DFLUX #HETVAL #Thermal Simulation #Fortran