Abaqus User Subroutine

Milad Vahidian, Ph.D. Candidate of Mechanical Engineering

Course Outline

Abaqus/Standard User Subroutines

1. DISP
2. DLOAD
3. UTRACLOAD
4. UTEMP
5. FILM
6. DFLUX
7. UEXPAN
8. UAMP
9. SIGINI
10. UFILD
11. USDFLD
12. UVARM
13. UMAT
14. UHYPER
15. UELMAT
16. UEL

Course Outline

Abaqus/Explicit User Subroutines

1. VDISP
2. VDLOAD
3. VUTEMP
4. VDFLUX
5. VUEXPAN
6. VUAMP
7. VUFILD
8. VUSDFLD
9. VUVARM
10. VUMAT
11. VUHYPER
12. VUEL

Reference

- Abaqus Documentation
- Writing User Subroutines with ABAQUS
- SIMULIA Documentation

Linking Abaqus & FORTRAN

User Subroutine

1- Abaqus/CAE 2022

↓

(CAE=Complete Abaqus Environment)

2-Microsoft Visual Studio 2019

3-Intel Parallel Studio 2020

Linking Abaqus & FORTRAN: Modifying Target

Step 1: Installing Abaqus/CAE, Visual Studio, and Intel Parallel Studio respectively.

Step 2: Modifying Target

Adding this address to “Abaqus Command”, “Abaqus Verification”, and “Abaqus CAE” target

“C:\Program Files (x86)\IntelSWTools\compilers_and_libraries_2020.4.311\windows\bin\ifortvars.bat” intel64 vs2019 &

Step 3: Verification

❑ Abaqus Verification: run Abaqus Verification and cheek the .log file out

❑ Abaqus Command: Enter “abaqus info=system” , “abaqus verify -user_std” and “abaqus verify -user_exp”

Linking Abaqus & FORTRAN: Modifying abq2022

Step 1: Installing Abaqus/CAE, Visual Studio, and Intel Parallel Studio respectively.

Step 2: Finding the directory of "**ifortvars.bat**", "**ifort.exe**", and "**vcvars64.bat**"

By default: C:\Program Files (x86)\IntelSWTools\compilers_and_libraries_2020.4.311\windows\bin

Step 3: Adding these variable and associated directory into "**Environment variables**"

Step 4: Modifying abq2022

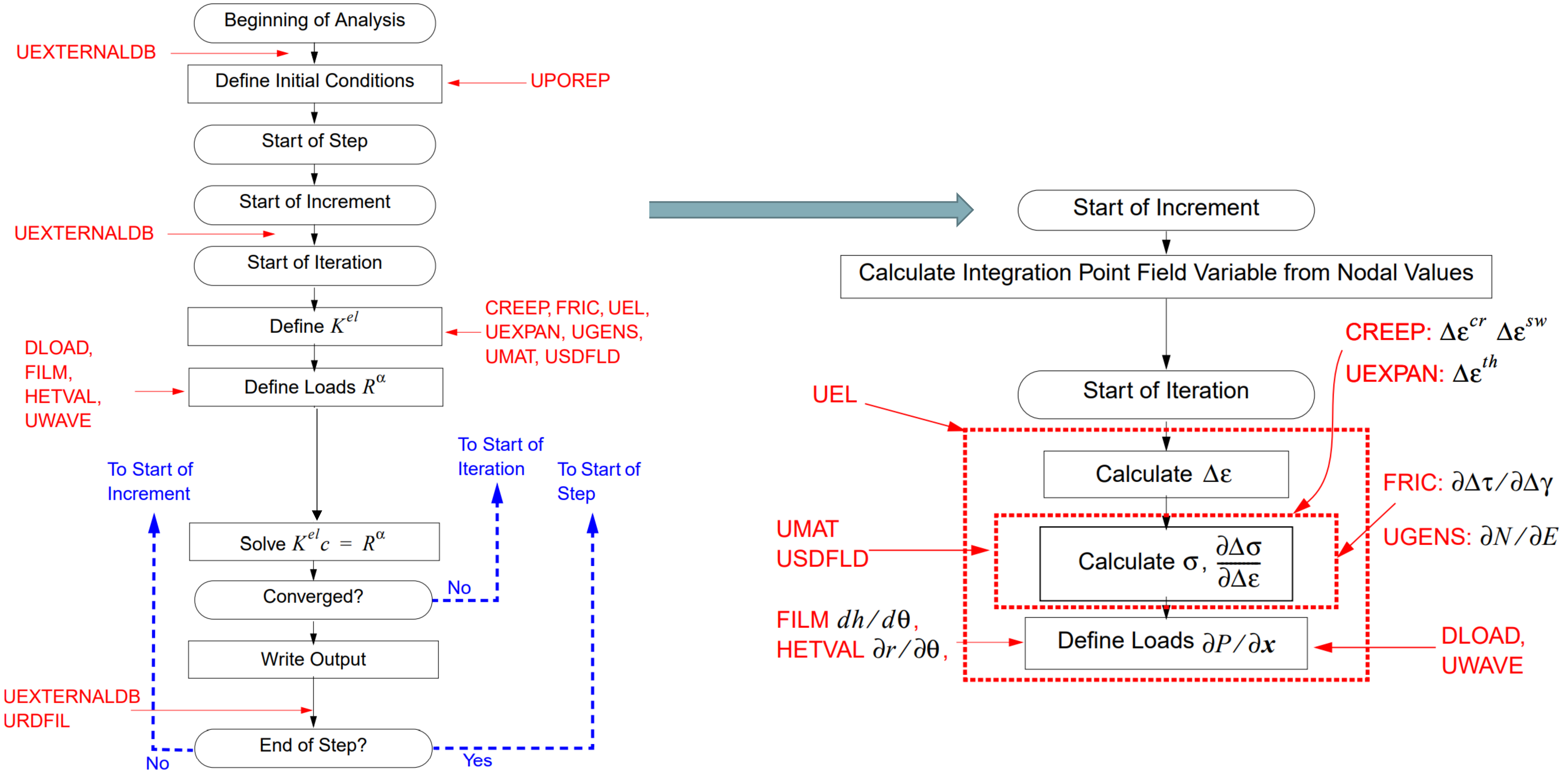
Adding this Code to abq2022.bat (By default) in **C:\SIMULIA\Commands**

```
@call ifortvars.bat intel64 vs2019
```

Step 5: Verification

- ❑ Abaqus Verification: run Abaqus Verification and cheek the .log file out
- ❑ Abaqus Command: Enter "abaqus info=system", "abaqus verify -user_std", and "abaqus verify -user_exp"

Where User Subroutines Fit into Abaqus/Standard



Beginning of Analysis

UEXTERNALDB

Define Initial Conditions

UPOREP

Start of Step

Start of Increment

UEXTERNALDB

Start of Iteration

Define K^{el}

CREEP, FRIC, UEL, UEXPAN, UGENS, UMAT, USDFLD

DLOAD, FILM, HETVAL, UWAVE

Define Loads R^{α}

To Start of Increment

To Start of Iteration

To Start of Step

Solve $K^{el}c = R^{\alpha}$

Converged?

No

Write Output

UEXTERNALDB URDFIL

End of Step?

No

Yes

Start of Increment

Calculate Integration Point Field Variable from Nodal Values

CREEP: $\Delta\varepsilon^{cr}$ $\Delta\varepsilon^{sw}$

UEXPAN: $\Delta\varepsilon^{th}$

UEL

Start of Iteration

Calculate $\Delta\varepsilon$

FRIC: $\partial\Delta\tau/\partial\Delta\gamma$

UGENS: $\partial N/\partial E$

UMAT USDFLD

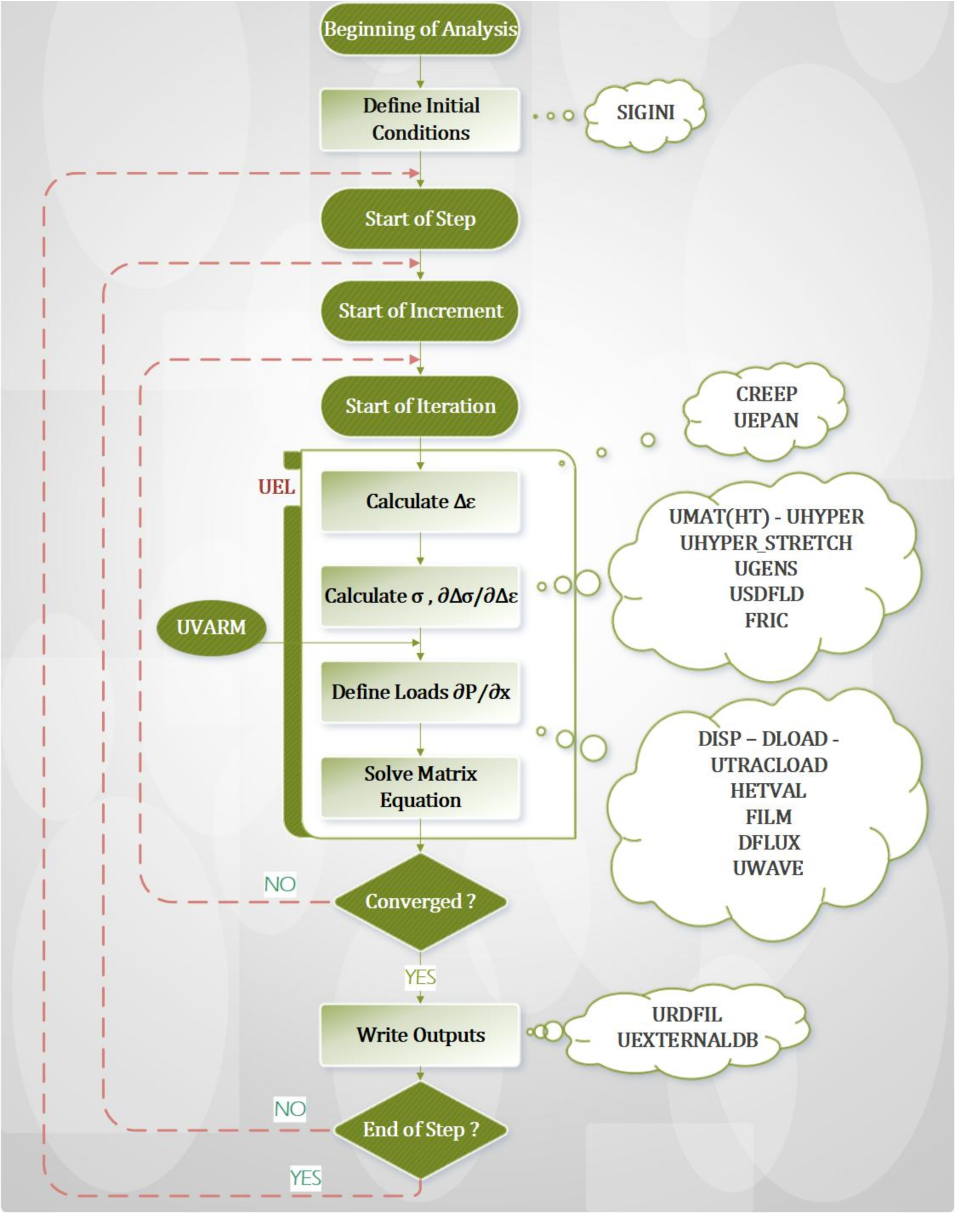
Calculate σ, $\dfrac{\partial\Delta\sigma}{\partial\Delta\varepsilon}$

FILM $dh/d\theta$, HETVAL $\partial r/\partial\theta$,

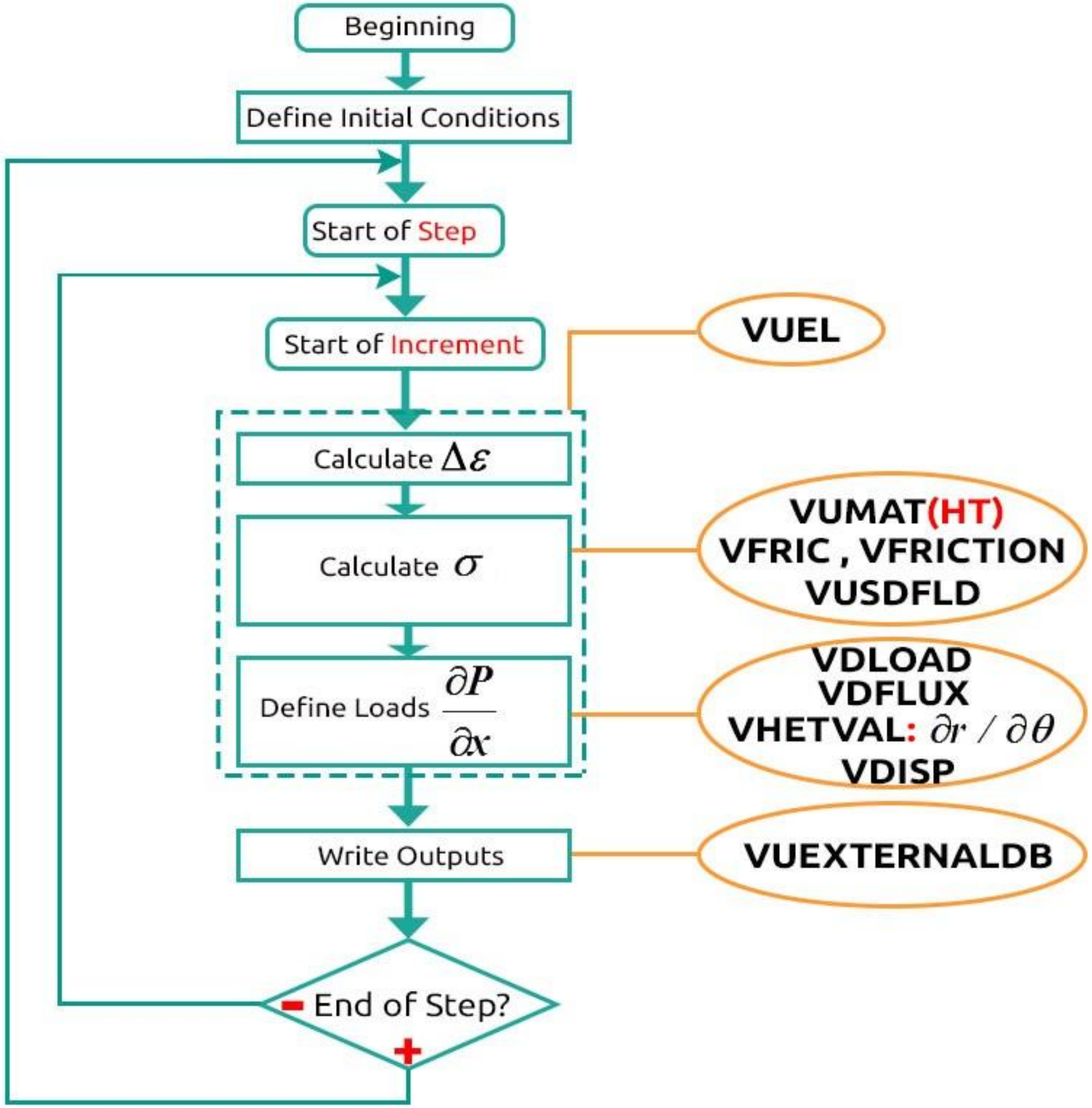
Define Loads $\partial P/\partial \mathbf{x}$

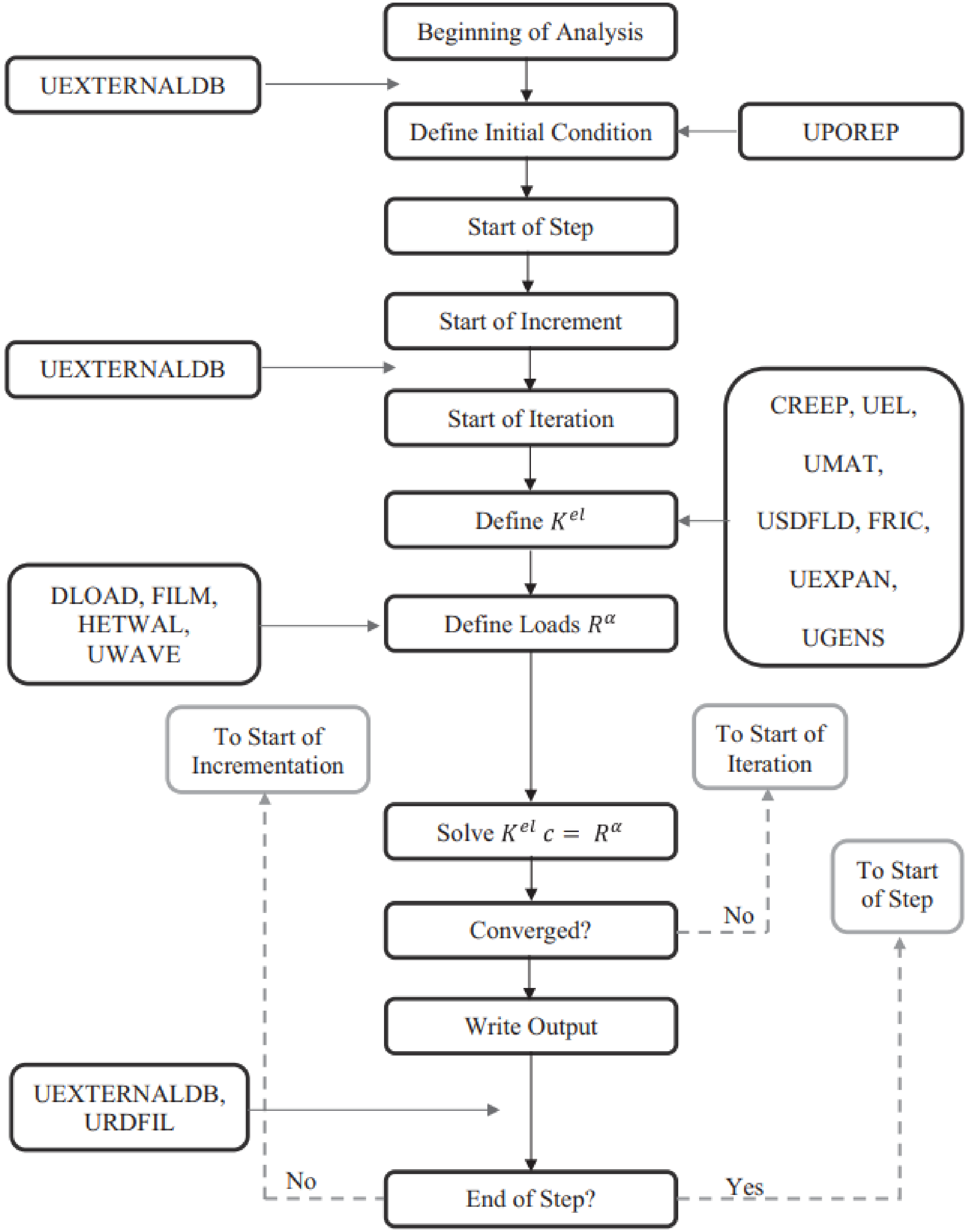
DLOAD, UWAVE

Where User Subroutines Fit into Abaqus/Standard



Where User Subroutines Fit into Abaqus/Explicit





Beginning of Analysis

UEXTERNALDB

Define Initial Condition

UPOREP

Start of Step

Start of Increment

UEXTERNALDB

Start of Iteration

CREEP, UEL,

UMAT,

USDFLD, FRIC,

UEXPAN,

UGENS

Define K^{el}

DLOAD, FILM, HETWAL, UWAVE

Define Loads R^{α}

To Start of Incrementation

To Start of Iteration

Solve $K^{el}\,c = R^{\alpha}$

To Start of Step

Converged?

No

Write Output

UEXTERNALDB, URDFIL

No

End of Step?

Yes

Some Tips

- User Subroutines are written as C, C++, or **Fortran** code

- In The First iteration of an increment all of user subroutines are called twice

  During the first call the initial stiffness matrix is being formed using the configuration of the model at the start of the increment.

  During the second call a new stiffness, based on the updated configuration of the model, is created.

  In subsequent iterations the subroutines are called only once.

- In these subsequent iterations the corrections to the model's configuration are calculated using the stiffness from the end of the previous iteration.

Some Tips

- Using multiple user subroutines in a model

When multiple user subroutines are needed in the analysis, the individual routines can be combined into a single file.

A given user subroutine (such as UMAT or FILM) should appear only once in the specified user subroutine source or object code.

- Restart analyses

When an analysis that includes a user subroutine is restarted, the user subroutine must be specified again because the subroutine object or source code is not stored on the restart (. res) file.

Code	Unit Number	Description
Abaqus/Standard	1	Internal database
	2	Solver file
	6	Printed output (.dat) file (You can write output to this file.)
	7	Message (.msg) file (You can write output to this file.)
	8	Results (.fil) file
	10	Internal database
	12	Restart (.res) file
	19–30	Internal databases (scratch files). Unit numbers 21 and 22 are always written to disk.
	73	Text file containing meshed beam cross-section properties (.bsp)

Code	Unit Number	Description
Abaqus/Explicit	6	Printed output (.log)
	12	Restart (.res) file
	13	Old restart (.res) file, if applicable
	15	Analysis Preprocessor (.dat or .pre) file
	23	Communications (.023) file
	60	Global package (.pac) file
	61	Global state (.abq) file
	62	Temporary file
	63	Global selected results (.sel) file
	64	Message (.msg) file
	65	Output database (.odb) file
	67	Old package (.pac) file, if import from Abaqus/Explicit
	68	Old state (.abq) file, if import from Abaqus/Explicit
	69	Internal database; temporary file
If domain-parallel	70	Local package (.pac.1) file for CPU #1
	71	Local state (.abq.1) file for CPU #1
	73	Local selected results (.sel.1) file for CPU #1
	80	Local package (.pac.2) file for CPU #2
	81	Local state (.abq.2) file for CPU #2
	83	Local selected results (.sel.2) file for CPU #2
	...	Add three files, incrementing units by 10, for each additional CPU

Some Tips

- The following unit numbers can be used within a user subroutine to read and write data from files:
  - 15-18
  - 100+

- In Abaqus/Standard user subroutines can write debug output to:
  - Log File (.log) → Unit *
  - Message File (.msg) → Unit 7
  - Print Output File (.dat) → Unit 6

These units do not have to be opened within the user subroutine— they are opened by Abaqus.

- In Abaqus/Explicit user subroutines can write debug output to the message
  - Log File (.log) → Unit *
  - Write to the status (.sta) → Unit 6

Some Tips

- Path names for external files

When a file is opened in a user subroutine, Abaqus assumes that it is located in the **scratch directory** created for the simulation.

Therefore, **full path names** must be used in the OPEN statements in the subroutine to specify the location of the files.

The following example opens, reads and closes an external file:

```
     open(unit=15, file=‘/nfs_scratch/wdir/ndw/TempHist.inp’)

     read(15,*) (timehist(j), j=1,25

     i = 1
     do while ( .true. )
        read(15,*,end=100) index(i),(temphist(i,j), j=1,25)
        i = i + 1
     end do

100 close(15)
```

Some Tips

- Every user subroutine in Abaqus/Standard must include the statement:

```
include 'aba_param.inc'
```

As the first statement after the argument list

The file specifies implicit real*8 (a-h, o-z) for double precision machines

The Abaqus execution procedure, which compiles and links the user subroutine with the rest of Abaqus, will include the aba_param.inc file automatically.

- It is not necessary to find this file and copy it to any particular directory: Abaqus will know where to find it

- Every user subroutine in Abaqus/ Explicit must include the statement

```
include 'vaba_param.inc'
```

Some Tips

- **Naming conventions**

  If user subroutines call other subroutines or use COMMON blocks to pass information, the names of such subroutines or COMMON blocks should begin with the letter K since this letter is never used to start the name of any **subroutine** or COMMON block in Abaqus.

- **Subroutine argument lists**

  - The variables passed into a user subroutine via the argument list are classified as either **variables to be defined**, **variables that can be updated**, or **variables passed in for information**.

  - The user must not alter the values of the "variables passed in for information." Doing so will yield unpredictable results.

Some Tips

➢ Solution-dependent state variables

- Solution-dependent state variables (SDVs) are values that can be defined to evolve with the solution. An example of a solution-dependent state variable for the UEL subroutine is strain.
- Several user subroutines allow the user to define SDVs.
- Within these user subroutines the SDVs can be defined as functions of any variables passed into the user subroutine.
- It is the user's responsibility to calculate the evolution of the SDVs within the subroutine; Abaqus just stores the variables for the user subroutine.
- For most subroutines the number of such variables required at the integration points or nodes is entered as the only value on the data line of the *DEPVAR option.
- For subroutines (V)UEL, UELMAT, and UGENS the VARIABLES parameter must be used on the *USER ELEMENT and *SHELL GENERAL SECTION options, as appropriate.
- For subroutine FRIC the number of variables is defined with the DEPVAR parameter on the *FRICTION option

Some Tips

➢ Solution-dependent state variables

- There are two methods available for defining the initial values of solution-dependent variables.
- The *INITIAL CONDITIONS, TYPE=SOLUTION option can be used to define the variable field in a tabular format
- For complicated cases user subroutine SDVINI can be used to define the initial values of the SDVs (Abaqus/Standard only).
- Invoke this subroutine by adding the USER parameter to the *INITIAL CONDITIONS, TYPE=SOLUTION option.

DepVar: In Property

SDV: In Field Output

STATEV: In UMAT

NSTATV

Some Tips

- Testing suggestions

Always develop and test user subroutines on the smallest possible model.

Do not include other complicated features, such as contact, unless they are absolutely necessary when testing the subroutine.

Test the most basic variant of the user subroutine before adding any new features to it.

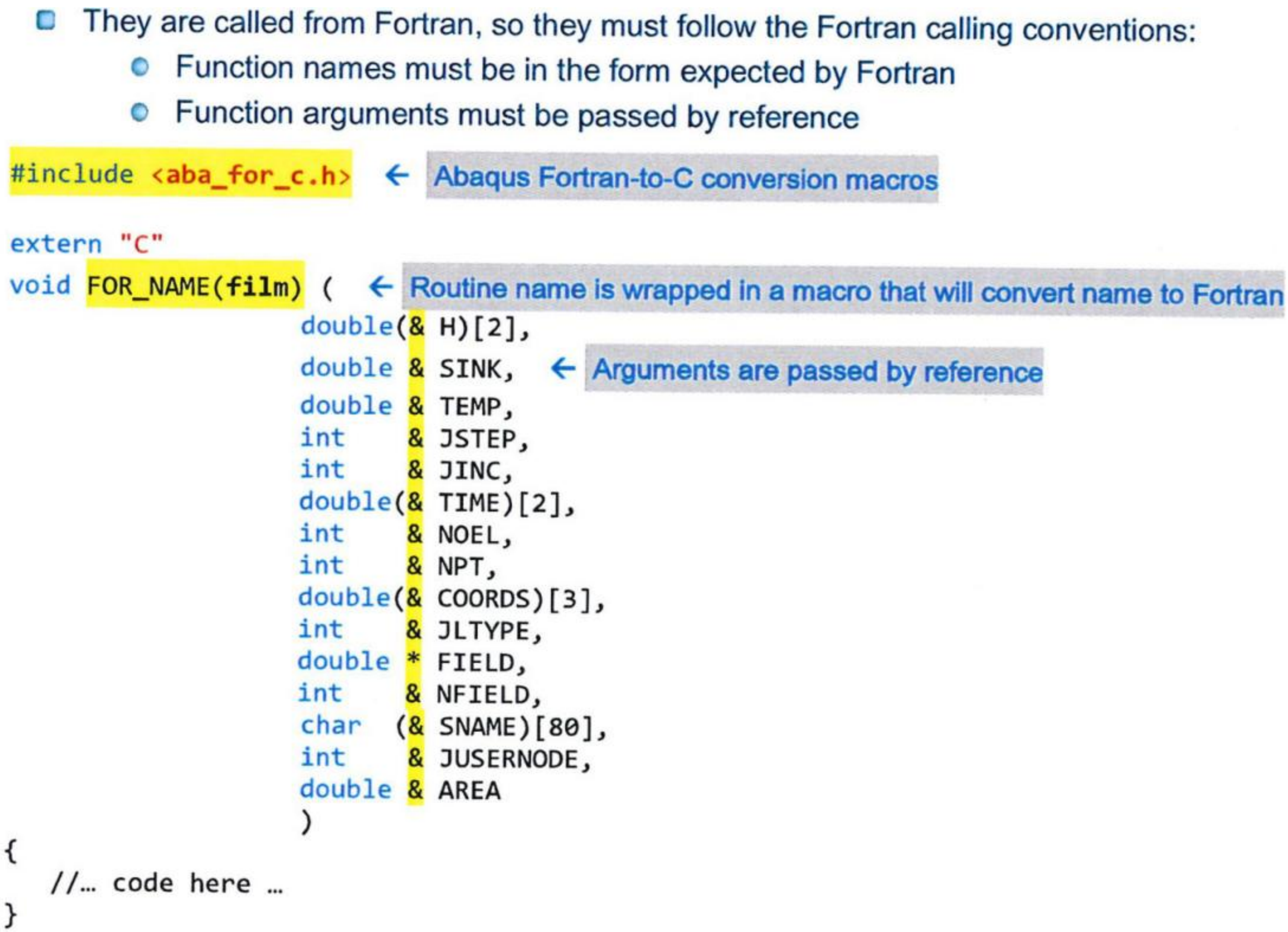
When appropriate, try to test the user subroutine with models where only values of the nodal degrees of freedom (displacement, rotations, temperature) are specified.

Then test the subroutine with models where fluxes and nodal degrees of freedom are specified.

Ensure that arrays passed into a user subroutine with a given dimension are not used as if they had a larger dimension. For example, if a user subroutine is written such that the number of SDVs is 10 but only 8 SDVs are specified on the *DEPVAR option, the user subroutine will overwrite data stored by Abaqus with unpredictable consequences.

Some Tips

- User subroutines may also be written in C or C++



- They are called from Fortran, so they must follow the Fortran calling conventions:
  - Function names must be in the form expected by Fortran
  - Function arguments must be passed by reference

```
#include <aba_for_c.h>   ← Abaqus Fortran-to-C conversion macros

extern "C"
void FOR_NAME(film) (   ← Routine name is wrapped in a macro that will convert name to Fortran
                  double(& H)[2],
                  double & SINK,   ← Arguments are passed by reference
                  double & TEMP,
                  int    & JSTEP,
                  int    & JINC,
                  double(& TIME)[2],
                  int    & NOEL,
                  int    & NPT,
                  double(& COORDS)[3],
                  int    & JLTYPE,
                  double * FIELD,
                  int    & NFIELD,
                  char  (& SNAME)[80],
                  int    & JUSERNODE,
                  double & AREA
                  )
{
   //… code here …
}
```

An Introduction to Fortran

Fortran, as derived from **Formula Translating System**, is a general-purpose, imperative programming language. It is used for numeric and scientific computing

Fortran was originally developed by IBM in the 1950s for scientific and engineering applications. Fortran ruled this programming area for a long time and became very popular for high performance computing

- Numerical analysis and scientific computation
- Structured programming
- Array programming
- Modular programming
- Generic programming
- High performance computing on supercomputers
- Object oriented programming
- Concurrent programming
- Reasonable degree of portability between computer systems

An Introduction to Fortran

Fortran is case-insensitive, except for string literals.

```
program program_name
implicit none

! type declaration statements
! executable statements

end program program_name
```

The implicit none statement allows the compiler to check that all your variable types are declared properly. You must always use implicit none at the start of every program.

Fortran Keywords

The non-I/O keywords				
allocatable	allocate	assign	assignment	block data
call	case	character	common	complex
contains	continue	cycle	data	deallocate
default	do	double precision	else	else if
elsewhere	end block data	end do	end function	end if
end interface	end module	end program	end select	end subroutine
end type	end where	entry	equivalence	exit
external	function	go to	if	implicit
in	inout	integer	intent	interface
intrinsic	kind	len	logical	module
namelist	nullify	only	operator	optional
out	parameter	pause	pointer	private
program	public	real	recursive	result
return	save	select case	stop	subroutine
target	then	type	type()	use

Fortran Keywords

The I/O related keywords				
backspace	close	endfile	format	inquire
open	print	read	rewind	Write

Fortran Intrinsic Data Types

Integer type

```
integer(kind = 2) :: integer_var
```

Real type

```
real :: real_var

real :: real_var
```

Complex type

```
complex :: complex_var

complex_var = cmplx (2.0, -7.0)
```

Logical type

```
logical :: logical_var

logical_var = .true.
```

Character type

```
character(len = 40) :: name

name = “Hello World”
```

Constants

Fixed Values That The Program Cannot Alter During Its Execution

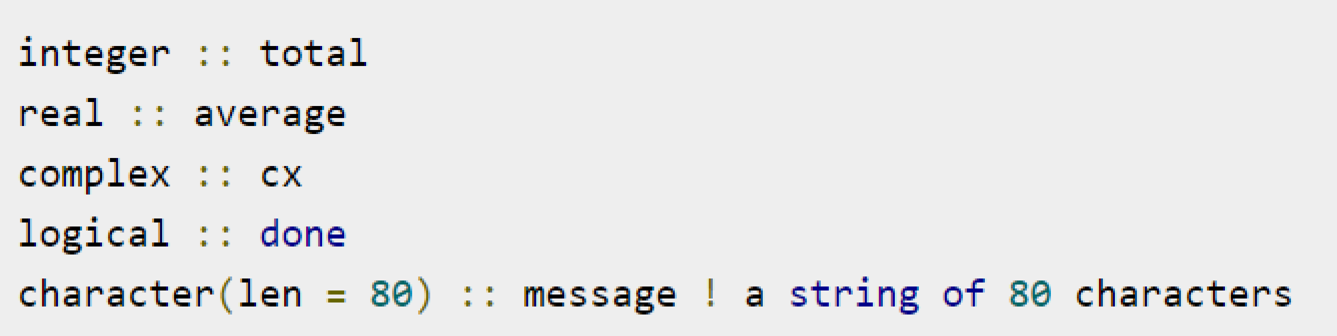
```
real, parameter :: pi = 3.1415927
```

Variable Declaration

Variables are declared at the beginning of a program (or subprogram) in a type declaration statement.

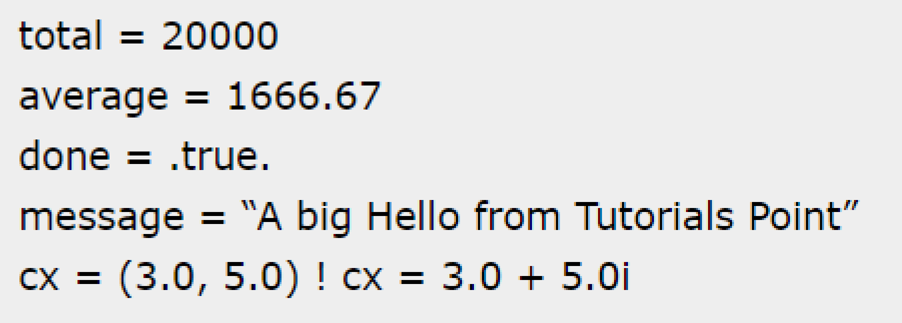
Syntax

```
type-specifier :: variable_name
```



```
integer :: total
real :: average
complex :: cx
logical :: done
character(len = 80) :: message ! a string of 80 characters
```

Later you can assign values to these variables, like,



```
total = 20000
average = 1666.67
done = .true.
message = “A big Hello from Tutorials Point”
cx = (3.0, 5.0) ! cx = 3.0 + 5.0i
```

Arithmetic Operators

Operator	Description	Example
+	Addition Operator, adds two operands.	A + B will give 8
-	Subtraction Operator, subtracts second operand from the first.	A - B will give 2
*	Multiplication Operator, multiplies both operands.	A * B will give 15
/	Division Operator, divides numerator by de-numerator.	A / B will give 1
**	Exponentiation Operator, raises one operand to the power of the other.	A ** B will give 125

Relational Operators

Operator	Equivalent	Description	Example
==	.eq.	Checks if the values of two operands are equal or not, if yes then condition becomes true.	(A == B) is not true.
/=	.ne.	Checks if the values of two operands are equal or not, if values are not equal then condition becomes true.	(A != B) is true.
>	.gt.	Checks if the value of left operand is greater than the value of right operand, if yes then condition becomes true.	(A > B) is not true.
<	.lt.	Checks if the value of left operand is less than the value of right operand, if yes then condition becomes true.	(A < B) is true.
>=	.ge.	Checks if the value of left operand is greater than or equal to the value of right operand, if yes then condition becomes true.	(A >= B) is not true.
<=	.le.	Checks if the value of left operand is less than or equal to the value of right operand, if yes then condition becomes true.	(A <= B) is true.

Logical Operators

Operator	Description	Example
.and.	Called Logical AND operator. If both the operands are non-zero, then condition becomes true.	(A .and. B) is false.
.or.	Called Logical OR Operator. If any of the two operands is non-zero, then condition becomes true.	(A .or. B) is true.
.not.	Called Logical NOT Operator. Use to reverses the logical state of its operand. If a condition is true then Logical NOT operator will make false.	!(A .and. B) is true.
.eqv.	Called Logical EQUIVALENT Operator. Used to check equivalence of two logical values.	(A .eqv. B) is false.
.neqv.	Called Logical NON-EQUIVALENT Operator. Used to check non-equivalence of two logical values.	(A .neqv. B) is true.

Decisions

Sr. No	Statement & Description
1	If… then construct An **if… then… end if** statement consists of a logical expression followed by one or more statements.
2	If… then...else construct An **if… then** statement can be followed by an optional **else statement,** which executes when the logical expression is false.
3	if...else if...else Statement An **if** statement construct can have one or more optional **else-if** constructs. When the **if** condition fails, the immediately followed **else-if** is executed. When the **else-if** also fails, its successor **else-if** statement (if any) is executed, and so on.
4	nested if construct You can use one **if** or **else if** statement inside another **if** or **else if** statement(s).
5	select case construct A **select case** statement allows a variable to be tested for equality against a list of values.
6	nested select case construct You can use one **select case** statement inside another **select case** statement(s).

Decisions

Loops

Sr. No	Loop Type & Description
1	do loop This construct enables a statement, or a series of statements, to be carried out iteratively, while a given condition is true.
2	do while loop Repeats a statement or group of statements while a given condition is true. It tests the condition before executing the loop body.
3	nested loops You can use one or more loop construct inside any other loop construct.

Sr. No	Control Statement & Description
1	exit If the exit statement is executed, the loop is exited, and the execution of the program continues at the first executable statement after the end do statement.
2	cycle If a cycle statement is executed, the program continues at the start of the next iteration.
3	stop If you wish execution of your program to stop, you can insert a stop statement

Loops

Characters

Character Declaration

```
type-specifier :: variable_name
```

→

```
character(len = 15) :: surname, firstname
```

len(string): It returns the length of a character string

index(string, sustring): It finds the location of a substring in another string, returns 0 if not found.

achar(int): It converts an integer into a character

iachar(c): It converts a character into an integer

trim(string): It returns the string with the trailing blanks removed.

scan(string, chars): It searches the "string" from left to right (unless back=.true.) for the first occurrence of any character contained in "chars". It returns an integer giving the position of that character, or zero if none of the characters in "chars" have been found.

verify(string, chars): It scans the "string" from left to right (unless back=.true.) for the first occurrence of any character not contained in "chars". It returns an integer giving the position of that character, or zero if only the characters in "chars" have been found

adjustl(string): It left justifies characters contained in the "string"

adjustr(string): It right justifies characters contained in the "string"

len_trim(string): It returns an integer equal to the length of "string" (len(string)) minus the number of trailing blanks

repeat(string, ncopy): It returns a string with length equal to "ncopy" times the length of "string", and containing "ncopy" concatenated copies of "string"

lle(char, char): Compares whether the first character is lexically less than or equal to the second

lge(char, char): Compares whether the first character is lexically greater than or equal to the second

lgt(char, char): Compares whether the first character is lexically greater than the second

llt(char, char): Compares whether the first character is lexically less than the second

Arrays

Declaring Arrays

```
real, dimension(5) :: numbers

integer, dimension (5,5) :: matrix

real, dimension(2:6) :: numbers

integer, dimension (-3:2,0:4) :: matrix
```

Assigning Values

```
numbers(1) = 2.0

Do i  = 1,5
   numbers(i) = i * 2.0
End Do

numbers = (/1.5, 3.2, 4.5, 0.9, 7.2/)
```

Array Sections

```
array ([lower]:[upper])

array ([lower]: )

array ( :[upper])

array ([lower]:[upper][:stride], ...)
```

```
B(2:10) = (/1.5, 3.2, 3.6, 4.5, 5.4, 6.8, 0.9, 7.2/)

B(2:) = (/1.5, 3.2, 3.6, 4.5, 5.4, 6.8, 0.9, 7.2/)

B(:8) = (/1.5, 3.2, 3.6, 4.5, 5.4, 6.8, 0.9, 7.2/)

B(2:10:2) = (/1.5, 3.2, 4.5, 0.9, 7.2/)
B(2:10:2) = [1.5, 3.2, 4.5, 0.9, 7.2]
```

Arrays

Rank	It is the number of dimensions an array has. For example, for the array named matrix, rank is 2, and for the array named numbers, rank is 1.
Extent	It is the number of elements along a dimension. For example, the array numbers has extent 5 and the array named matrix has extent 3 in both dimensions.
Shape	The shape of an array is a one-dimensional integer array, containing the number of elements (the extent) in each dimension. For example, for the array matrix, shape is (3, 3) and the array numbers it is (5).
Size	It is the number of elements an array contains. For the array matrix, it is 9, and for the array numbers, it is 5.

Vector and matrix multiplication

Function	Description
`dot_product(vector_a, vector_b)`	This function returns a scalar product of two input vectors, which must have the same length.
`matmul(matrix_a, matrix_b)`	It returns the matrix product of two matrices, which must be consistent, i.e. have the dimensions like (m, k) and (k, n)

Reduction Functions

Function	Description
`all(mask, dim)`	It returns a logical value that indicates whether all relations in mask are .true., along with only the desired dimension if the second argument is given.
`any(mask, dim)`	It returns a logical value that indicates whether any relation in mask is .true., along with only the desired dimension if the second argument is given.
`count(mask, dim)`	It returns a numerical value that is the number of relations in mask which are .true., along with only the desired dimension if the second argument is given.
`maxval(array, dim, mask)`	It returns the largest value in the array array, of those that obey the relation in the third argument mask, if that one is given, along with only the desired dimension if the second argument dim is given.
`minval(array, dim, mask)`	It returns the smallest value in the array array, of those that obey the relation in the third argument mask, if that one is given, along with only the desired dimension if the second argument DIM is given.
`product(array, dim, mask)`	It returns the product of all the elements in the array array, of those that obey the relation in the third argument mask, if that one is given, along with only the desired dimension if the second argument dim is given.
`sum(array, dim, mask)`	It returns the sum of all the elements in the array array, of those that obey the relation in the third argument mask, if that one is given, along with only the desired dimension if the second argument dim is given.

Inquiry Functions

Function & Description
`allocated(array)` It is a logical function which indicates if the array is allocated.
`lbound(array, dim)` It returns the lower dimension limit for the array. If dim (the dimension) is not given as an argument, you get an integer vector, if dim is included, you get the integer value with exactly that lower dimension limit, for which you asked.
`shape(source)` It returns the shape of an array source as an integer vector.
`size(array, dim)` It returns the number of elements in an array. If dim is not given, and the number of elements in the relevant dimension if dim is included.
`ubound(array, dim)` It returns the upper dimensional limits.

Construction Functions

Function	Description
`merge(tsource, fsource, mask)`	This function joins two arrays. It gives the elements in tsource if the condition in mask is .true. and fsource if the condition in mask is .false. The two fields tsource and fsource have to be of the same type and the same shape. The result also is of this type and shape. Also, mask must have the same shape.
`pack(array, mask, vector)`	It packs an array to a vector with the control of mask. The shape of the logical array mask, has to agree with the one for array, or else mask must be a scalar. If vector is included, it has to be an array of rank 1 (i.e. a vector) with at least as many elements as those that are true in mask, and have the same type as array. If mask is a scalar with the value .true. then vector instead must have the same number of elements as array.
`spread(source, dim, ncopies)`	It returns an array of the same type as the argument source with the rank increased by one. The parameters dim and ncopies are integer. if ncopies is negative the value zero is used instead. If source is a scalar, then spread becomes a vector with ncopies elements that all have the same value as source. The parameter dim indicates which index is to be extended. it has to be within the range 1 and 1+(rank of source), if source is a scalar then dim has to be one. The parameter ncopies is the number of elements in the new dimensions.
`unpack(vector, mask, array)`	It scatters a vector to an array under control of mask. The shape of the logical array mask has to agree with the one for array. The array vector has to have the rank 1 (i.e. it is a vector) with at least as many elements as those that are true in mask, and also has to have the same type as array. If array is given as a scalar then it is considered to be an array with the same shape as mask and the same scalar elements everywhere. The result will be an array with the same shape as mask and the same type as vector. The values will be those from vector that are accepted, while in the remaining positions in array the old values are kept.

Reshape Functions

Function	Description
`reshape(source, shape, pad, order)`	It constructs an array with a specified shape starting from the elements in a given array source. If pad is not included then the size of source has to be at least product (shape). If pad is included, it has to have the same type as source. If order is included, it has to be an integer array with the same shape as shape and the values must be a permutation of (1,2,3,...,n), where n is the number of elements in shape , it has to be less than, or equal to 7.

Manipulation Functions

Function	Description
`cshift(array, shift, dim)`	It performs circular shift by shift positions to the left, if shift is positive and to the right if it is negative. If array is a vector the shift is being done in a natural way, if it is an array of a higher rank then the shift is in all sections along the dimension dim. If dim is missing it is considered to be 1, in other cases it has to be a scalar integer number between 1 and n (where n equals the rank of array). The argument shift is a scalar integer or an integer array of rank n-1 and the same shape as the array, except along the dimension dim (which is removed because of the lower rank). Different sections can therefore be shifted in various directions and with various numbers of positions.
`eoshift(array, shift, boundary, dim)`	It is end-off shift. It performs shift to the left if shift is positive and to the right if it is negative. Instead of the elements shifted out new elements are taken from boundary. If array is a vector the shift is being done in a natural way, if it is an array of a higher rank, the shift on all sections is along the dimension dim. if dim is missing, it is considered to be 1, in other cases it has to have a scalar integer value between 1 and n (where n equals the rank of array). The argument shift is a scalar integer if array has rank 1, in the other case it can be a scalar integer or an integer array of rank n-1 and with the same shape as the array except along the dimension dim (which is removed because of the lower rank).
`transpose (matrix)`	It transposes a matrix, which is an array of rank 2. It replaces the rows and columns in the matrix.

Location Functions

Function	Description
maxloc(array, mask)	It returns the position of the greatest element in the array, if mask is included only for those which fulfil the conditions in mask, position is returned and the result is an integer vector.
minloc(array, mask)	It returns the position of the smallest element in the array, if mask is included only for those which fulfil the conditions in mask, position is returned and the result is an integer vector.

Basic Input Output

```
read(*,*) item1, item2, item3...
print *,  item1, item2, item3
write(*,*) item1, item2, item3...
```

Formatted Input Output

```
read  fmt, variable_list
print fmt, variable_list
write fmt, variable_list
```

format specification

Procedures

A procedure is a group of statements that perform a well-defined task and can be invoked from your program. Information (or data) is passed to the calling program, to the procedure as arguments.

Functions

```
function name(arg1, arg2, ....)
   [declarations, including those for the arguments]
   [executable statements]
end function [name]
```

```
function name(arg1, arg2, ....)
   [declarations, including those for the arguments]
   [executable statements]
end function [name]
```

Subroutines

```
subroutine name(arg1, arg2, ....)
   [declarations, including those for the arguments]
   [executable statements]
end subroutine [name]
```

Numeric Functions

Function	Description
`abs(a)`	It returns the absolute value of A
`aimag(z)`	It returns the imaginary part of a complex number Z
`aint(a [, kind])`	It truncates fractional part of A towards zero, returning a real, whole number.
`anint(a [, kind])`	It returns a real value, the nearest integer or whole number.
`ceiling(a [, kind])`	It returns the least integer greater than or equal to number A.
`cmplx(x [, y, kind])`	It converts the real variables X and Y to a complex number X + iY; if Y is absent, 0 is used.
`conjg(z)`	It returns the complex conjugate of any complex number Z.
`dble(a)`	It converts A to a double precision real number.
`dim(x, y)`	It returns the positive difference of X and Y.
`dprod(x, y)`	It returns the double precision real product of X and Y.
`floor(a [, kind])`	It provides the greatest integer less than or equal to number A.
`int(a [, kind])`	It converts a number (real or integer) to integer, truncating the real part towards zero.
`max(a1, a2 [, a3,...])`	It returns the maximum value from the arguments, all being of same type.
`min(a1, a2 [, a3,...])`	It returns the minimum value from the arguments, all being of same type.
`mod(a, p)`	It returns the remainder of A on division by P, both arguments being of the same type (A-INT(A/P)*P)
`modulo(a, p)`	It returns A modulo P: (A-FLOOR(A/P)*P)
`nint(a [, kind])`	It returns the nearest integer of number A
`real(a [, kind])`	It Converts to real type
`sign(a, b)`	It returns the absolute value of A multiplied by the sign of P. Basically it transfers the of sign of B to A.

Mathematical Functions

Function	Description
acos(x)	It returns the inverse cosine in the range (0, π), in radians.
asin(x)	It returns the inverse sine in the range (-π/2, π/2), in radians.
atan(x)	It returns the inverse tangent in the range (-π/2, π/2), in radians.
atan2(y, x)	It returns the inverse tangent in the range (-π, π), in radians.
cos(x)	It returns the cosine of argument in radians.
cosh(x)	It returns the hyperbolic cosine of argument in radians.
exp(x)	It returns the exponential value of X.
log(x)	It returns the natural logarithmic value of X.
log10(x)	It returns the common logarithmic (base 10) value of X.
sin(x)	It returns the sine of argument in radians.
sinh(x)	It returns the hyperbolic sine of argument in radians.
sqrt(x)	It returns square root of X.
tan(x)	It returns the tangent of argument in radians.
tanh(x)	It returns the hyperbolic tangent of argument in radians.

Numeric Inquiry Functions

Function	Description
`digits(x)`	It returns the number of significant digits of the model.
`epsilon(x)`	It returns the number that is almost negligible compared to one. In other words, it returns the smallest value such that REAL(1.0, KIND(X)) + EPSILON(X) is not equal to REAL(1.0, KIND(X)).
`huge(x)`	It returns the largest number of the model
`maxexponent(x)`	It returns the maximum exponent of the model
`minexponent(x)`	It returns the minimum exponent of the model
`precision(x)`	It returns the decimal precision
`radix(x)`	It returns the base of the model
`range(x)`	It returns the decimal exponent range
`tiny(x)`	It returns the smallest positive number of the model

Floating-Point Manipulation Functions

Function	Description
`exponent(x)`	It returns the exponent part of a model number
`fraction(x)`	It returns the fractional part of a number
`nearest(x, s)`	It returns the nearest different processor number in given direction
`rrspacing(x)`	It returns the reciprocal of the relative spacing of model numbers near given number
`scale(x, i)`	It multiplies a real by its base to an integer power
`set_exponent(x, i)`	it returns the exponent part of a number
`spacing(x)`	It returns the absolute spacing of model numbers near given number

Bit Manipulation Functions

Function	Description
`bit_size(i)`	It returns the number of bits of the model
`btest(i, pos)`	Bit testing
`iand(i, j)`	Logical AND
`ibclr(i, pos)`	Clear bit
`ibits(i, pos, len)`	Bit extraction
`ibset(i, pos)`	Set bit
`ieor(i, j)`	Exclusive OR
`ior(i, j)`	Inclusive OR
`ishft(i, shift)`	Logical shift
`ishftc(i, shift [, size])`	Circular shift
`not(i)`	Logical complement

Character Functions

Function	Description
`achar(i)`	It returns the Ith character in the ASCII collating sequence.
`adjustl(string)`	It adjusts string left by removing any leading blanks and inserting trailing blanks
`adjustr(string)`	It adjusts string right by removing trailing blanks and inserting leading blanks.
`char(i [, kind])`	It returns the Ith character in the machine specific collating sequence
`iachar(c)`	It returns the position of the character in the ASCII collating sequence.
`ichar(c)`	It returns the position of the character in the machine (processor) specific collating sequence.
`index(string, substring [, back])`	It returns the leftmost (rightmost if BACK is .TRUE.) starting position of SUBSTRING within STRING.
`len(string)`	It returns the length of a string.
`len_trim(string)`	It returns the length of a string without trailing blank characters.
`lge(string_a, string_b)`	Lexically greater than or equal
`lgt(string_a, string_b)`	Lexically greater than
`lle(string_a, string_b)`	Lexically less than or equal
`llt(string_a, string_b)`	Lexically less than
`repeat(string, ncopies)`	Repeated concatenation
`scan(string, set [, back])`	It returns the index of the leftmost (rightmost if BACK is .TRUE.) character of STRING that belong to SET, or 0 if none belong.
`trim(string)`	Removes trailing blank characters
`verify(string, set [, back])`	Verifies the set of characters in a string

Kind & Logical Functions

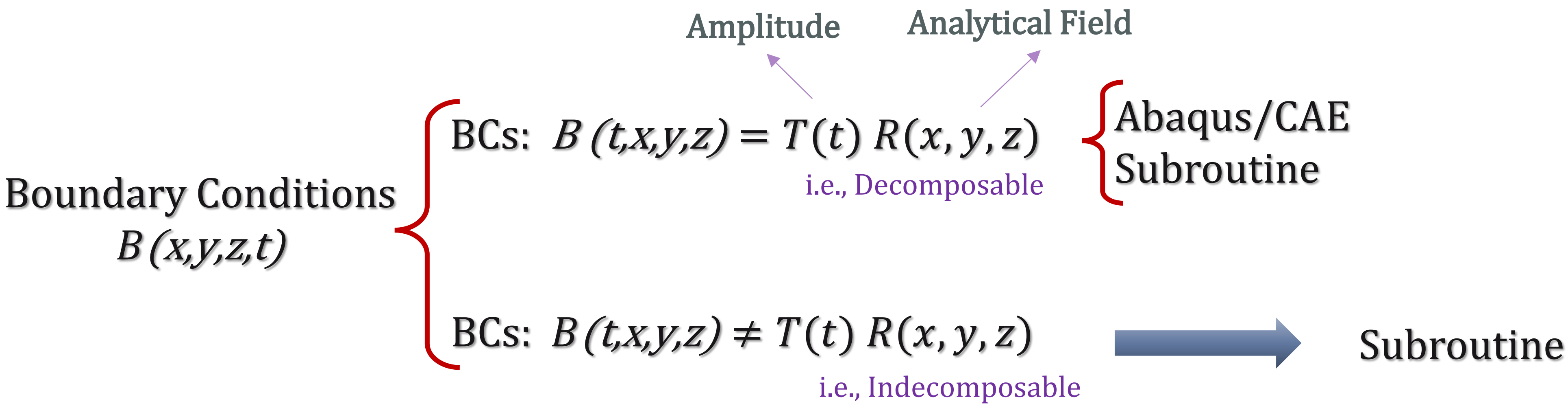
Function	Description
`kind (x)`	It returns the kind type parameter value.
`selected_int_kind (r)`	It returns kind of type parameter for specified exponent range.
`selected_real_kind ([p, r])`	Real kind type parameter value, given precision and range.
`logical (l [, kind])`	Convert between objects of type logical with different kind type parameters.

Program Libraries

RANDLIB, random number and statistical distribution generators
BLAS
EISPACK
GAMS–NIST Guide to Available Math Software
Some statistical and other routines from NIST
LAPACK
LINPACK
MINPACK
MUDPACK
NCAR Mathematical Library
The Netlib collection of mathematical software, papers, and databases.
ODEPACK
ODERPACK, a set of routines for ranking and ordering.
Expokit for computing matrix exponentials
SLATEC
SPECFUN
STARPAC
StatLib statistical library
TOMS
Sorting and merging strings

DISP

Abaqus User Subroutine To Specify Prescribed Boundary Conditions or Connectors Motion



Boundary Conditions $B\,(x,y,z,t)$

Amplitude

Analytical Field

BCs: $B\,(t,x,y,z) = \underbrace{T(t)}_{\text{Amplitude}}\ \underbrace{R(x,y,z)}_{\text{Analytical Field}}$

i.e., Decomposable

Abaqus/CAE
Subroutine

BCs: $B\,(t,x,y,z) \neq T(t)\ R(x,y,z)$

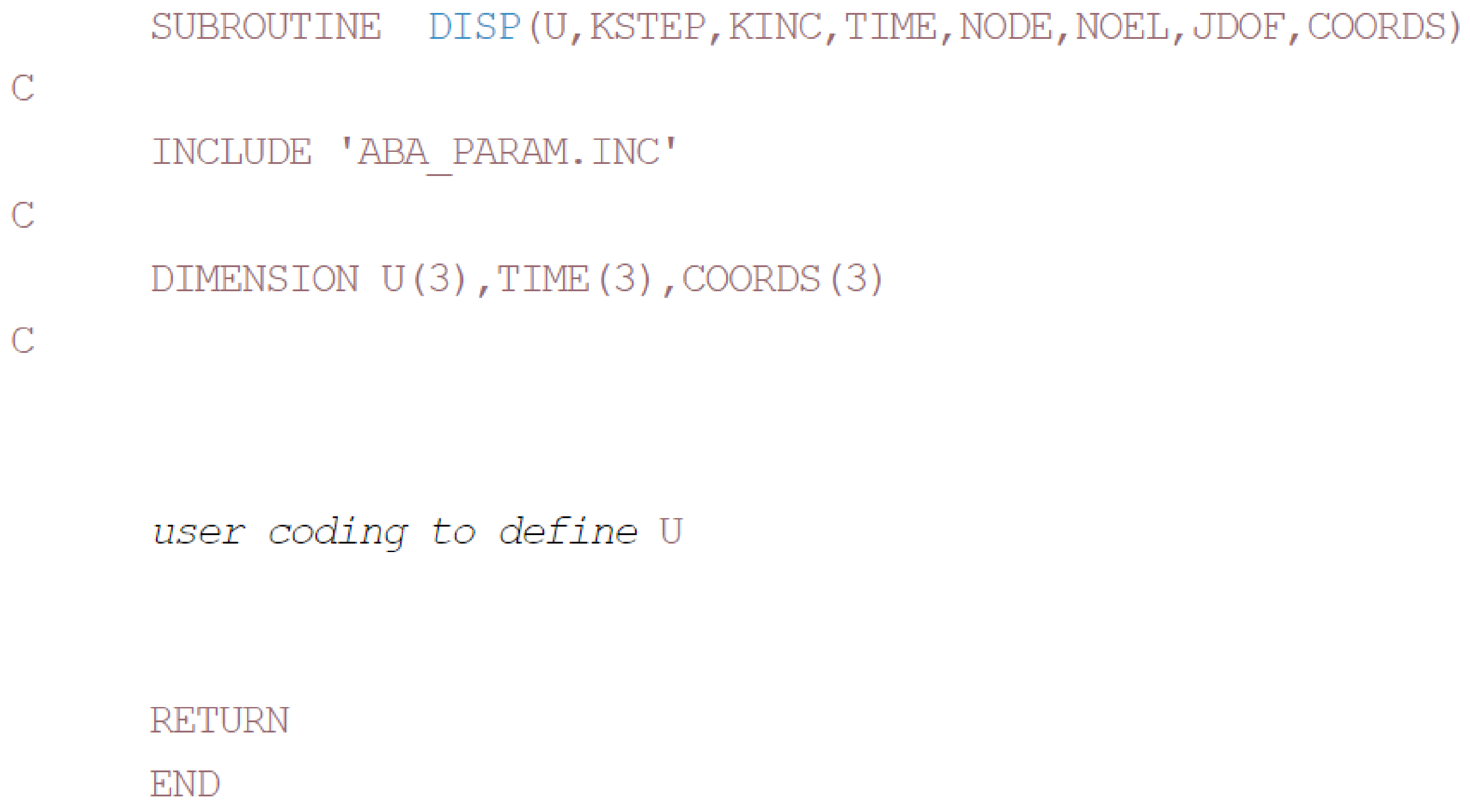
i.e., Indecomposable

Subroutine

DISP

Abaqus User Subroutine To Specify Prescribed Boundary Conditions or Connectors Motion

User Subroutine Interface



```
      SUBROUTINE  DISP(U,KSTEP,KINC,TIME,NODE,NOEL,JDOF,COORDS)
C
      INCLUDE 'ABA_PARAM.INC'
C
      DIMENSION U(3),TIME(3),COORDS(3)
C


      user coding to define U


      RETURN
      END
```

Variables to Be Defined

U

- U(1)
  - All variable types except rotation: the total value of the prescribed variable at this point.
  - Rotation variable type: the incremental value of the prescribed rotation at this point.
- $\mathrm{U}(2) = \frac{dU(1)}{dt}$ First Time Derivative of U(1)
- $\mathrm{U}(3) = \frac{d^2U(1)}{dt^2}$ Second Time Derivative of U(1)

Variables Passed in for Information

KSTEP		**Step number**	
KINC		**Increment number**	
TIME	TIME(1)	**Current value of step time**	
	TIME(2)	**Current value of total time**	
	TIME(3)	**Current value of time increment**	
NODE		**Node number**	→ This variable cannot be used if user subroutine DISP is used to prescribe connector motions.
NOEL		**Element number**	→ This variable cannot be used if user subroutine DISP is used to prescribe boundary conditions.
JDOF		**Degree of Freedom: NEXT SLIDE**	
COORDS		**An array containing the current coordinates of this point.**	This array cannot be used if user subroutine DISP is used to prescribe **connector motions.**

Degrees of freedom

1 *x*-displacement

2 *y*-displacement

3 *z*-displacement

4 Rotation about the *x*-axis, in radians

5 Rotation about the *y*-axis, in radians

6 Rotation about the *z*-axis, in radians

7 Warping amplitude (for open-section beam elements)

8 Pore pressure, hydrostatic fluid pressure, or acoustic pressure

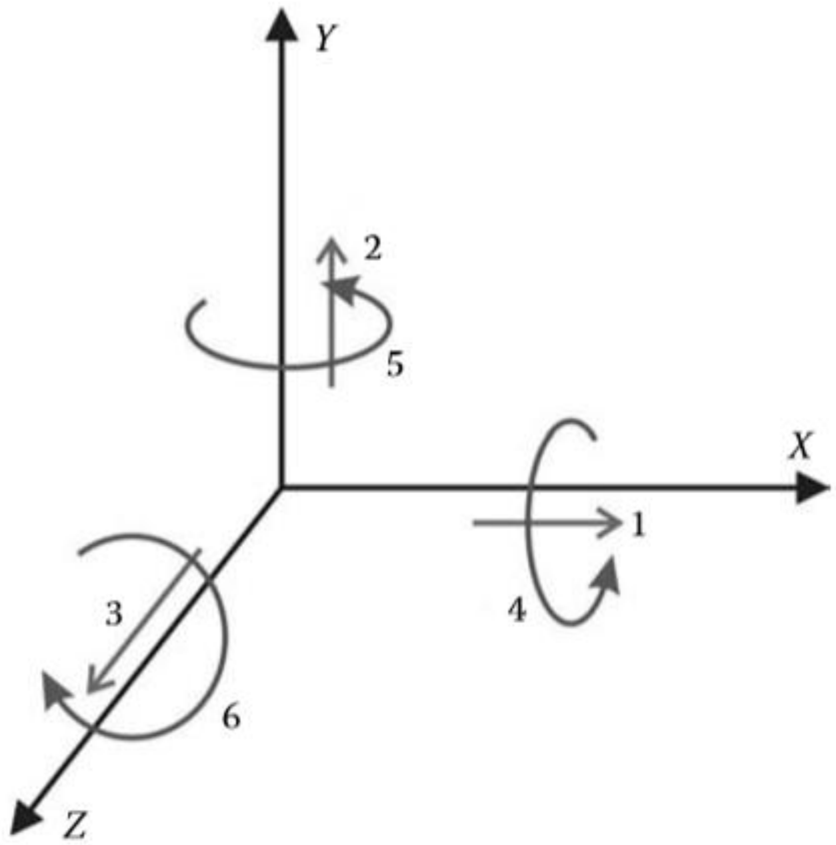
9 Electric potential

10 Connector material flow (units of length)

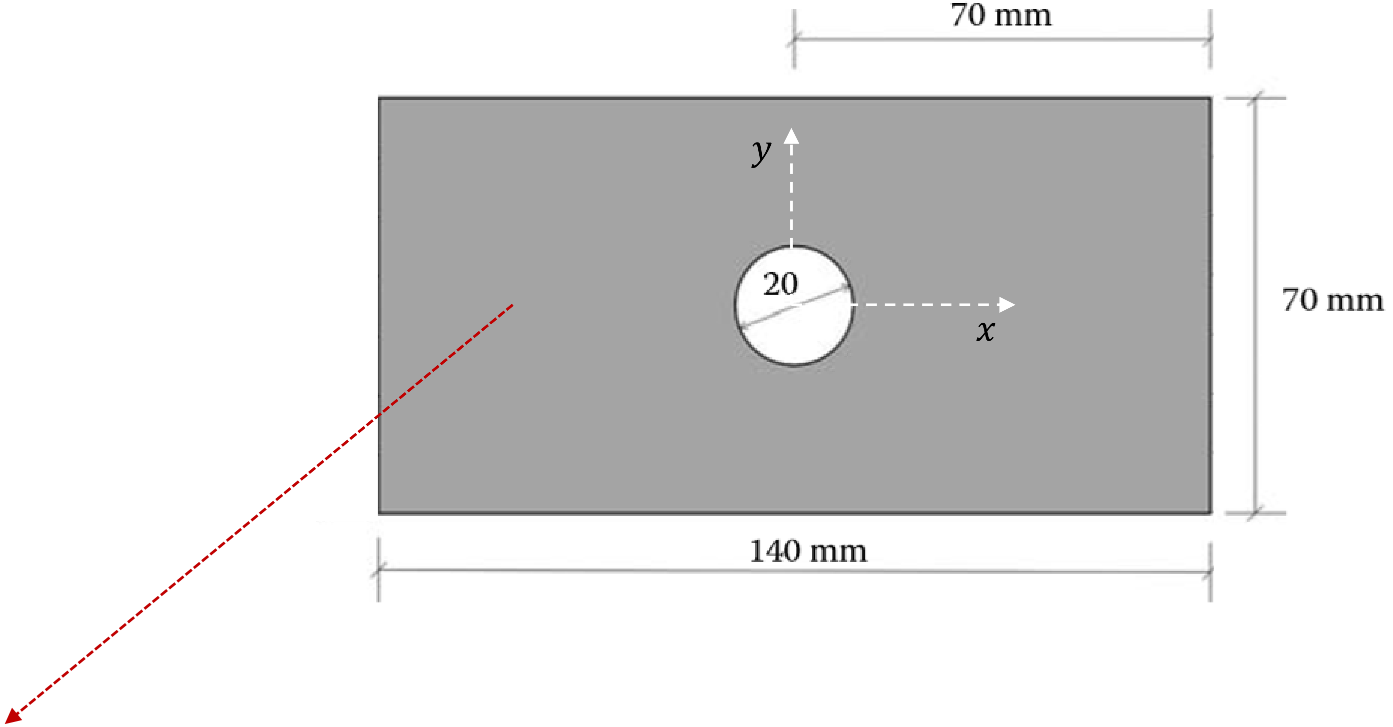
11 Temperature (or normalized concentration in mass diffusion analysis)

12 Second temperature (for shells or beams)

13 Third temperature (for shells or beams)



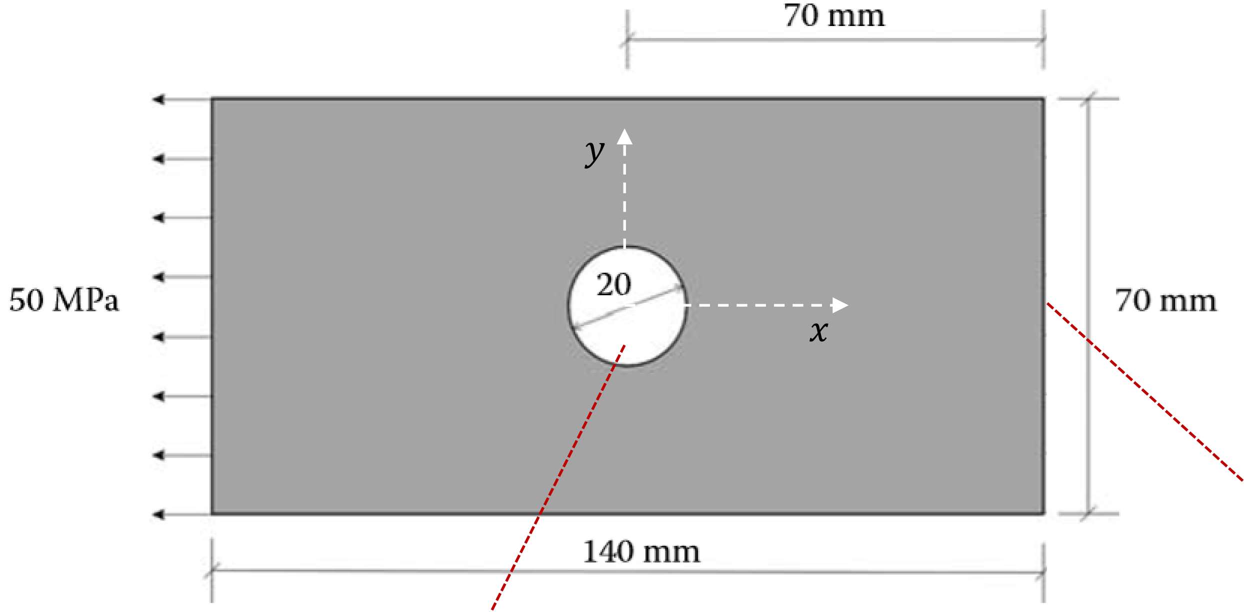
Disp Subroutine Problem



$$U_3 = 5\cos(10\pi t)\, sin\left(\frac{\pi x}{70}\right) sin\left(\frac{\pi y}{35}\right)$$

$$E = 70\ GPa \quad \nu = 0.33 \quad Thickness = 1\ mm$$

Disp Subroutine Problem

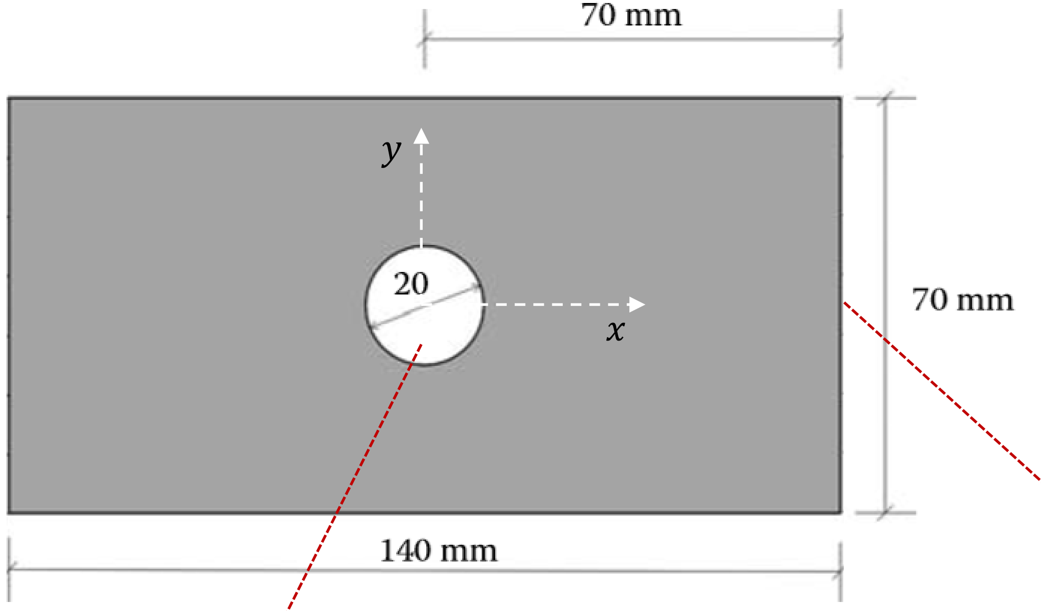


Fixed

$$U_1 = 5\,cos(10\pi t)\,sin\left(\frac{\pi y}{35}\right)$$

$E = 70\;GPa \quad \nu = 0.33 \quad Thickness = 1\;mm$

Disp Subroutine Problem



$$U_1 = -5\,cos(10\pi t)\,sin\left(\frac{\pi y}{35}\right)$$

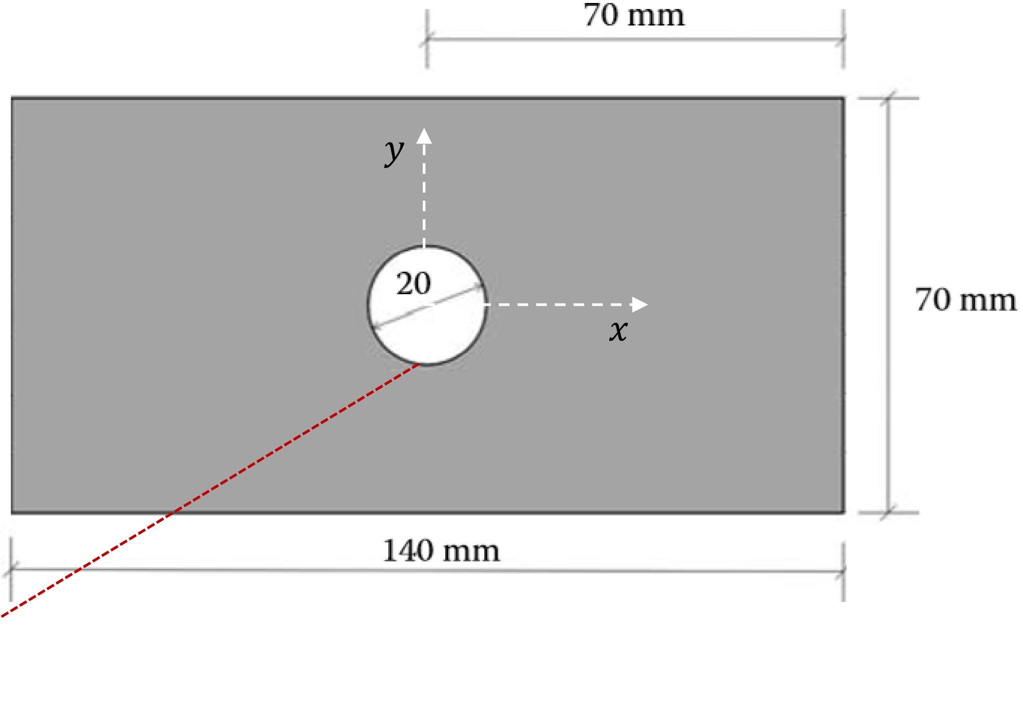
Fixed

$$U_1 = 5\,cos(10\pi t)\,sin\left(\frac{\pi y}{35}\right)$$

$E = 70\ GPa \qquad \nu = 0.33 \qquad Thickness = 1\ mm$

Disp Subroutine Problem

COORDS / NODE

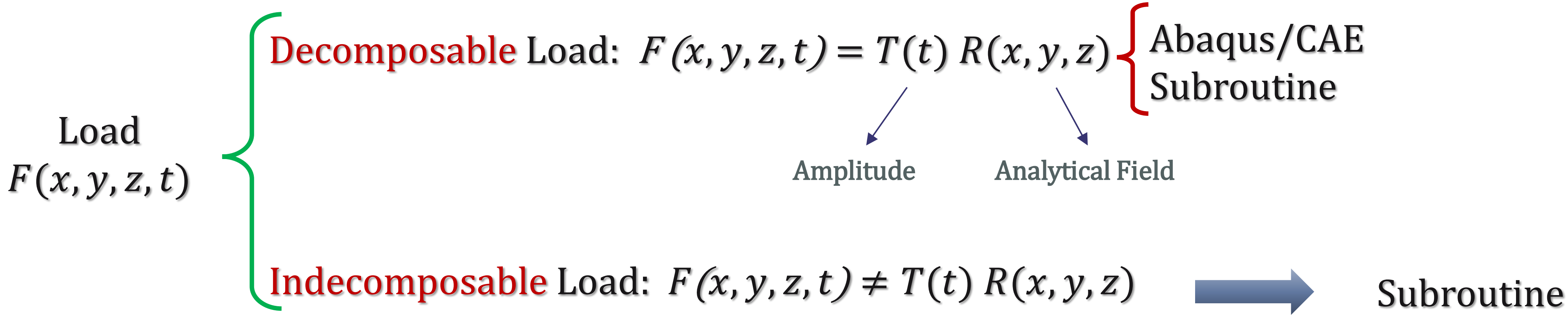


@ $x^2 + y^2 = 100$ ==> $U_3 = e^{-0.1t}$

$E = 70\ GPa \quad \nu = 0.33 \quad Thickness = 1\ mm$

DLOAD

Abaqus User Subroutine To Specify Non-uniform Distributed Load



Load $F(x,y,z,t)$

Decomposable Load: $F(x,y,z,t) = T(t)\,R(x,y,z)$

Abaqus/CAE

Subroutine

Amplitude

Analytical Field

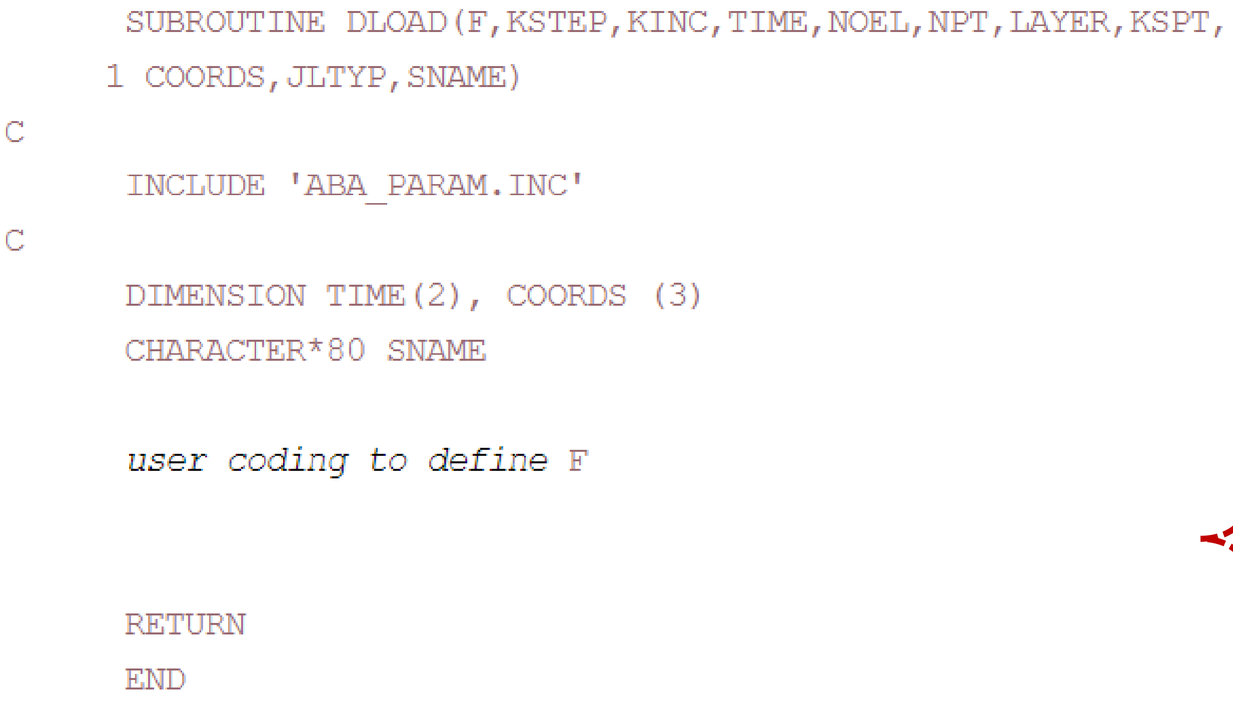
Indecomposable Load: $F(x,y,z,t) \neq T(t)\,R(x,y,z)$

Subroutine

The load is monitored by writing output to the printed output (.dat) file

DLOAD

Variables to be defined: **F**

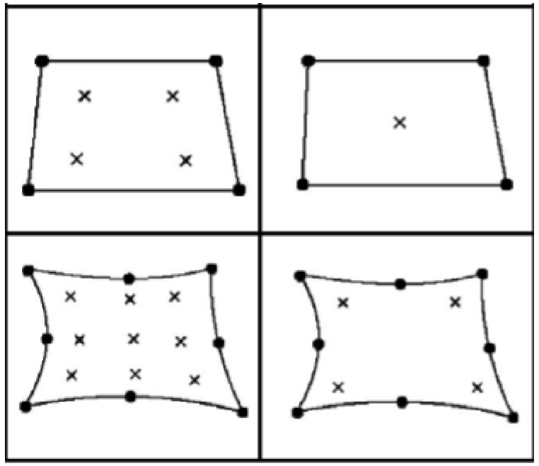


```
      SUBROUTINE DLOAD(F,KSTEP,KINC,TIME,NOEL,NPT,LAYER,KSPT,
     1 COORDS,JLTYP,SNAME)
C
      INCLUDE 'ABA_PARAM.INC'
C
      DIMENSION TIME(2), COORDS (3)
      CHARACTER*80 SNAME

      user coding to define F


      RETURN
      END
```

F		$\frac{F}{L^2}$ for surface loads and $\frac{F}{L^3}$ for body forces.
KSTEP		Step number
KINC		Increment number
TIME	TIME(1)	Current value of step time or current value of the load proportionality factor
	TIME(2)	Current value of total time
NOEL		Element number
NPT		Load integration point number within the element
LAYER		Layer number (for body forces in layered solids)
KSPT		Section point number within the current layer
COORDS		An array containing the coordinates of the load integration point
JLTYP		Load type
SNAME		Surface name for a surface-based load definition (JLTYP=0). For a body force or an element-based surface load the surface name is passed in as blank.

DLOAD

F		$\frac{F}{L}$ for Line loads, $\frac{F}{L^2}$ for surface loads, and $\frac{F}{L^3}$ for body forces.
KSTEP		Step number
KINC		Increment number
TIME	TIME(1)	Current value of step time or current value of the load proportionality factor λ, in a Riks step
	TIME(2)	Current value of total time
NOEL		Element number
NPT		Load integration point number within the element
LAYER		Layer number (for body forces in layered solids)
KSPT		Section point number within the current layer
COORDS		An array containing the coordinates of the load integration point. These are the current coordinates if geometric nonlinearity is accounted for during the step; otherwise, the array contains the original coordinates of the point.
JLTYP		Load type
SNAME		Surface name for a **surface-based** load definition (JLTYP=0). For a body force or an element-based surface load the surface name is passed in as blank.



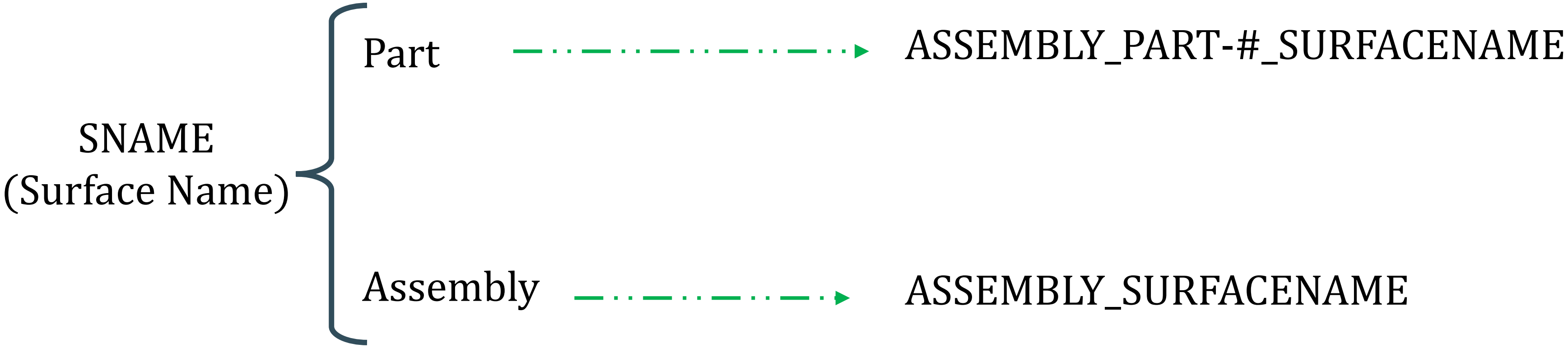
DLOAD

JLTYP	Load type	Description	Elements
0	Surface-based load		
1	BXNU	Nonuniform body force in global X-directions	
1	BRNU	Nonuniform body force in radial directions	
2	BYNU (except for axisymmetric elements)	Nonuniform body force in global Y-directions	
2	BZNU (for axisymmetric elements only)	Nonuniform body force in global Z-directions	
3	BZNU (for three-dimensional elements and **asymmetric-axisymmetric**)	Nonuniform body force in global Z-directions	
20	PNU	Nonuniform pressure	
21	P1NU	Nonuniform force per unit length in beam local 1-directions	Beam
22	P2NU	Nonuniform force per unit length in beam local 2-directions	Beam
23	P3NU		
24	P4NU		
25	P5NU		
26	P6NU		
27	PINU	Nonuniform internal pressure	PIPE & ELBOW
28	PENU	Nonuniform external pressure	PIPE & ELBOW
41	PXNU	Nonuniform force per unit length in global X-directions	Beam
42	PYNU	Nonuniform force per unit length in global Y-directions	Beam
43	PZNU	Nonuniform force per unit length in global Z-directions	Beam

DLOAD

SNAME

Surface name for a surface-based load definition (JLTYP=0). For a body force or an element-based surface load the surface name is passed in as blank.



SNAME
(Surface Name)

Part → ASSEMBLY_PART-#_SURFACENAME

Assembly → ASSEMBLY_SURFACENAME

DLOAD

Simulation time: 1 (s)

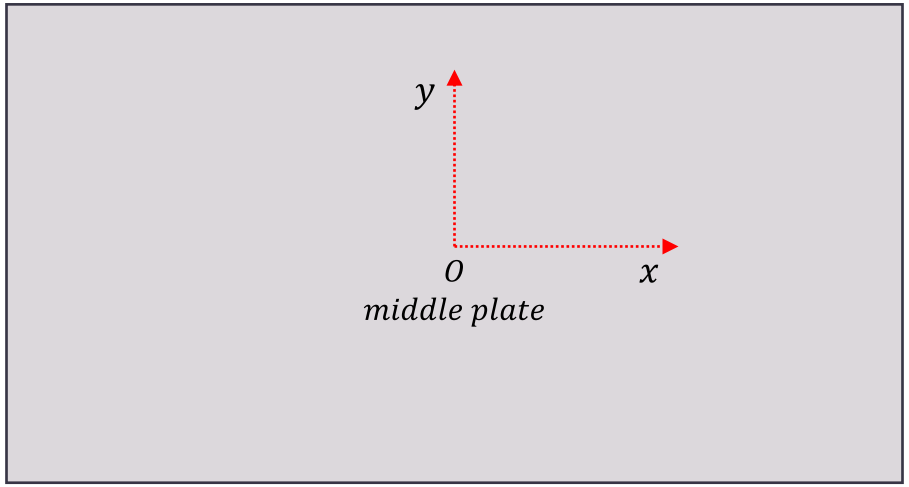
Hint: the JLTYP

Loads

- Surface Force (Pressure): exert on entire plate
- Body force: exert on whole plate

$$P(x,y,t) = cos(10\pi t)\, sin\left(\frac{\pi x}{300}\right) sin\left(\frac{\pi y}{200}\right)$$

$$F_b(x,y,t) = e^t\, sin\left(\frac{\pi x}{300}\right) \sin\left(\frac{\pi y}{200}\right)$$



All edge has been pinned

Plate's dimensions: 300x200 (mm), thickness: 2 (mm)

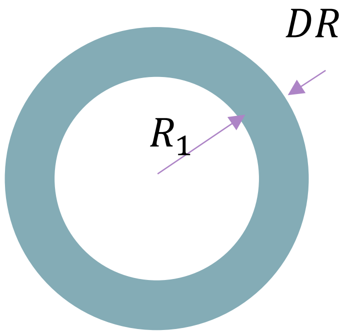
Material properties: E=200 GPa $\nu = 0.3$

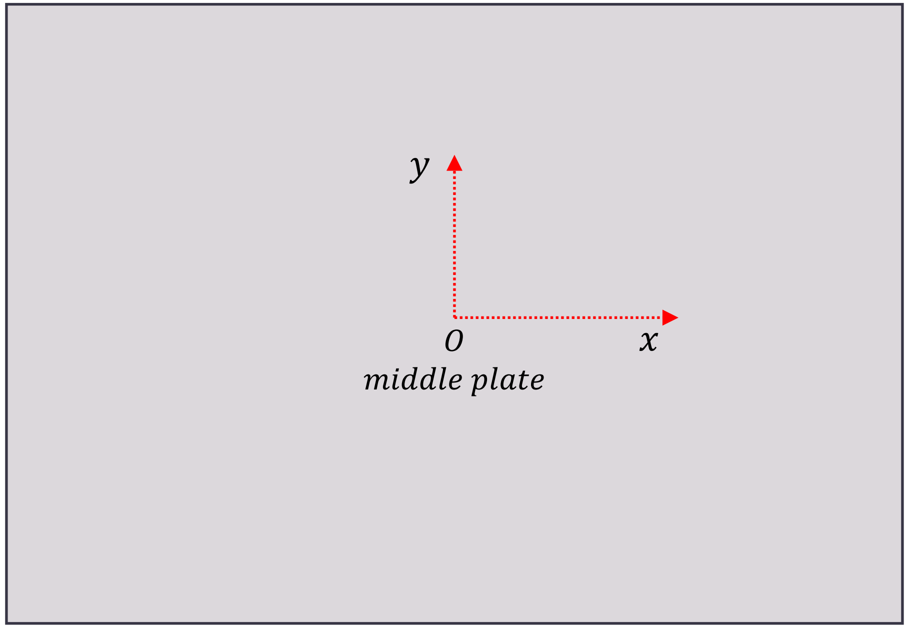
Dload: Moving Load

Moving Load:

Force reign is being changed by Time. → $Force\ reign = f(t)$

$$\begin{cases} r = \sqrt{x^2 + y^2} \\ \theta = \tan^{-1}\left(\frac{y}{x}\right) \end{cases}$$





$E = 200\ GPa,\ \ \nu = 0.3,\ \ \ dimension: 500 \times 500 \times 5$

Dload: Moving Load

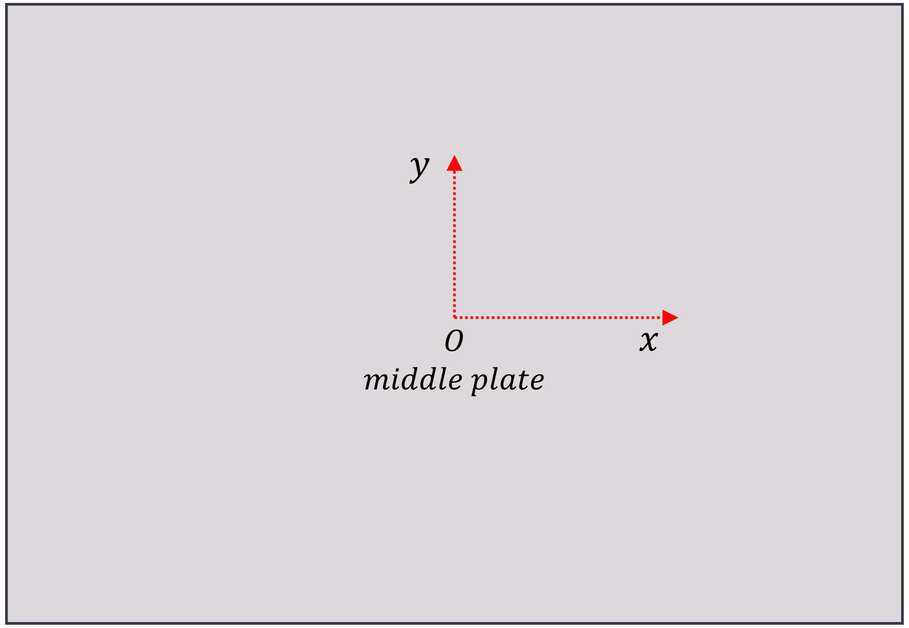
$$\mathbf{V} = \dot{r}\ \mathbf{e}_r + r\dot{\theta}\ \mathbf{e}_{\boldsymbol{\theta}}$$

$$\mathbf{e}_r = cos(\theta)\ \mathbf{e}_{\boldsymbol{x}} + sin(\theta)\ \mathbf{e}_y$$

$$\mathbf{e}_{\boldsymbol{\theta}} = -\,sin(\theta)\ \mathbf{e}_x + cos(\theta)\ \mathbf{e}_y$$

$$\begin{cases} \mathbf{V}_x = \dot{r}\ cos(\theta) - r\dot{\theta}\ sin(\theta) \\ \mathbf{V}_y = -\,\dot{r}\ sin(\theta) + r\dot{\theta}\ cos(\theta) \end{cases}$$

where
$$\begin{cases} r = \sqrt{x^2 + y^2} \\ \theta = \tan^{-1}\left(\dfrac{y}{x}\right) \end{cases}$$



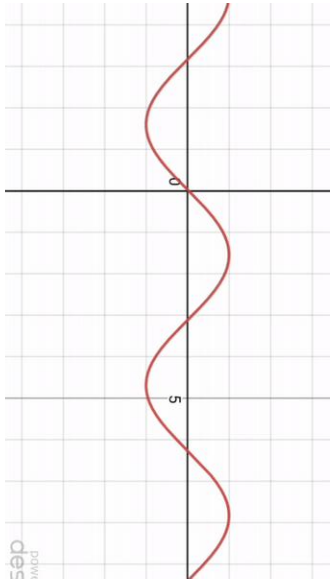
$$E = 200GPa, \quad \nu = 0.3, \qquad dimenstion: 500 \times 500 \times 5$$

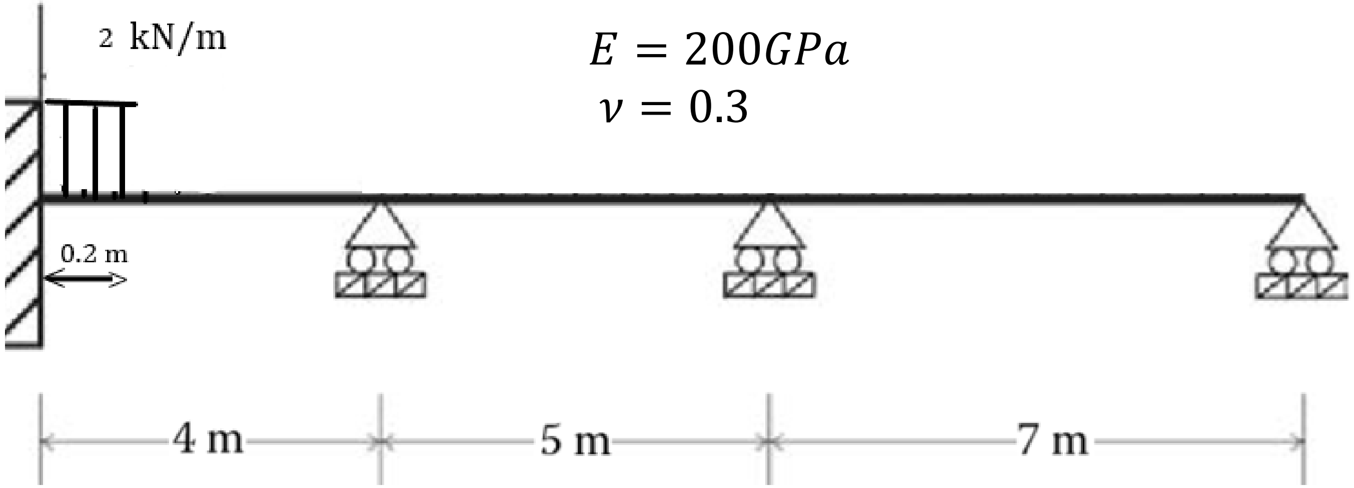
Dload: Periodic Travelling Wave

Body Load:

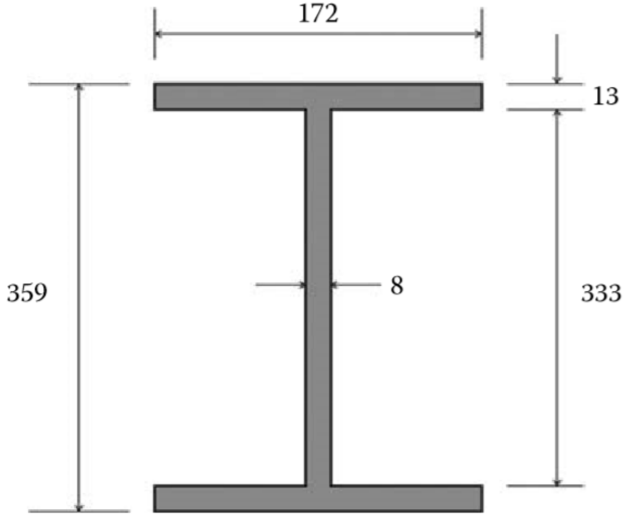
$$F_b(x,y,z,t) = cos(kz-\omega t)\, sin\left(\frac{\pi x}{300}\right) sin\left(\frac{\pi y}{200}\right)$$

$$k = \frac{2\pi}{\lambda} = 2\pi \qquad \omega = \frac{2\pi}{T} = \pi$$





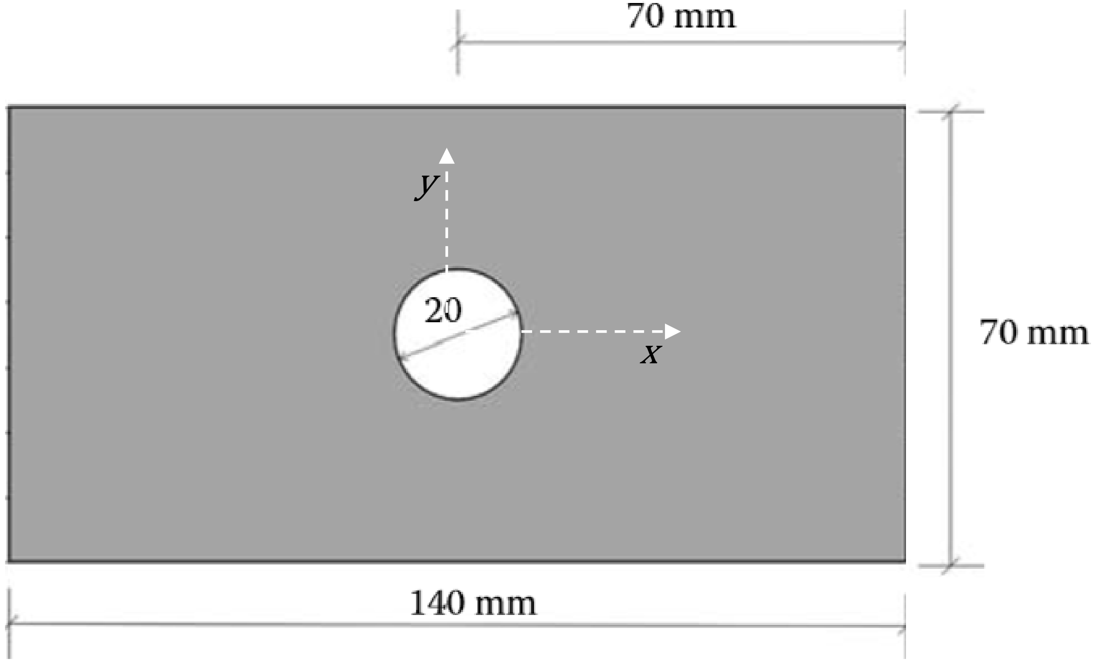
$E = 200GPa$
$\nu = 0.3$



Disp + Dload Subroutine

Simulation time: 1(s)

Material properties:
E=210 GPa $\nu = 0.3$ Thickness=2 mm



B.C's.

@ x=70 ==> $U_1 = 0, U_2 = 0, U_3 = \sin\left(\frac{\pi y}{35}\right)$

@ x=-70 ==>$U_1 = 0, U_2 = 0, U_3 = -\sin\left(\frac{\pi y}{35}\right)$

@ y=35 ==>$U_1 = 0, U_2 = 0, U_3 = \sin\left(\frac{\pi x}{70}\right)$

@ y=-35 ==>$U_1 = 0, U_2 = 0, U_3 = -\sin\left(\frac{\pi x}{70}\right)$

@ $x^2 + y^2 = 100$ ==> $U_3 = e^{-0.1t}$

Pressure

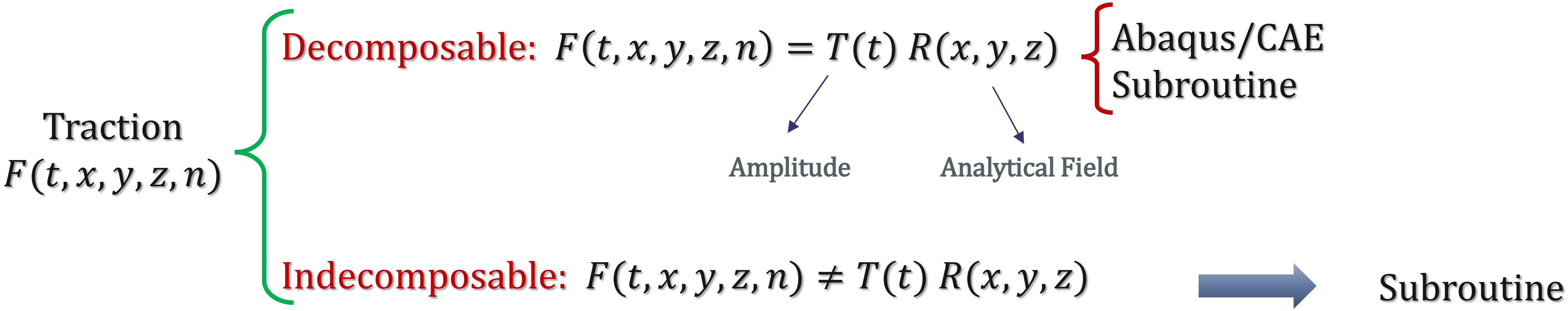
exert on right Up: $P(x, y, t) = \sin\left(\frac{\pi x}{70}\right) sin\left(\frac{\pi y}{35}\right) cos(10\pi t)$

exert on left Bottom: $P(x, y, t) = -\sin\left(\frac{\pi x}{70}\right) sin\left(\frac{\pi y}{35}\right) cos(10\pi t)$

Body Load

UTRACLOAD

Abaqus User Subroutine To Specify Non-uniform Traction Loads



Traction $F(t,x,y,z,n)$

Decomposable: $F(t,x,y,z,n) = T(t)\,R(x,y,z)$

Amplitude

Analytical Field

Abaqus/CAE

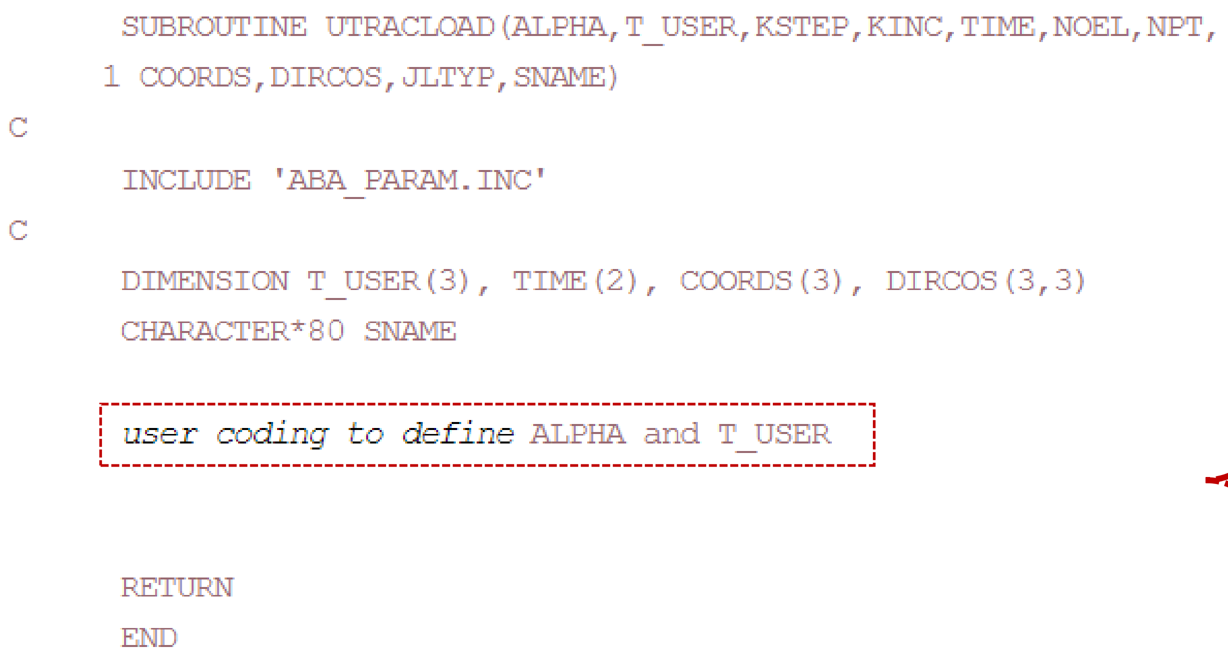
Subroutine

Indecomposable: $F(t,x,y,z,n) \neq T(t)\,R(x,y,z)$

Subroutine

UTRACLOAD

Abaqus User Subroutine To Specify Non-uniform Traction Loads



```
      SUBROUTINE UTRACLOAD(ALPHA,T_USER,KSTEP,KINC,TIME,NOEL,NPT,
     1 COORDS,DIRCOS,JLTYP,SNAME)
C
      INCLUDE 'ABA_PARAM.INC'
C
      DIMENSION T_USER(3), TIME(2), COORDS(3), DIRCOS(3,3)
      CHARACTER*80 SNAME

      user coding to define ALPHA and T_USER


      RETURN
      END
```

ALPHA		Magnitude of the distributed traction load
T_USER		Loading direction of the distributed traction load
KSTEP		Step number
KINC		Increment number
TIME	TIME(1)	Current value of step time or current value of the load proportionality factor
	TIME(2)	Current value of total time
NOEL		Element number
NPT		Load integration point number within the element
COORDS		An array containing the coordinates of the load integration point
DIRCOS		Orientation of the face or edge in the reference configuration
JLTYP		Identifies the load type

SNAME — Surface name for a **surface-based** load definition. For an element-based or edge-based load the surface name is passed in as blank

UTRACLOAD

ALPHA

Magnitude of the distributed traction load. Units are $\frac{F}{L^2}$ for surface loads, $\frac{F}{L}$ for edge loads, and F for edge moments. For a static analysis that uses the modified Riks method ALPHA must be defined as a function of the load proportionality factor, λ.

T_USER

Loading direction of the distributed traction load. The direction of T_USER should not change during a step. If it does, convergence difficulties might arise.

Load directions are needed
- General Surface Traction
- Shear Surface Traction
- General Edge Traction

Load directions will be ignored
- Edge Moment
- Shear Edge Traction
- Normal Edge Traction
- Transverse Edge Traction

COORDS

An array containing the coordinates of the **load integration point.** These are the current coordinates if geometric nonlinearity is accounted for during the step; otherwise, the array contains the original coordinates of the point.

DIRCOS

Orientation of the face or edge in the reference configuration. For three-dimensional facets the first and second columns are the normalized local directions **in the plane** of the surface, and **the third column is the normal** to the face. For solid elements the normal **points inward**; for shell elements the normal **points outward**. For two-dimensional facets the first column is the normalized tangent, the second column is the facet normal, and the third column is not used. For three-dimensional shell edges the first column is the tangent to the shell edge (**shear direction**), the second column is the in-plane normal (**normal direction**), and the third column is the normal to the plane of the shell (**transverse direction**).

UTRACLOAD

JLTYP → **Identifies the load type** for which this call to UTRACLOAD is being made.

This information is useful when several different nonuniform distributed loads are being imposed on an element at the same time

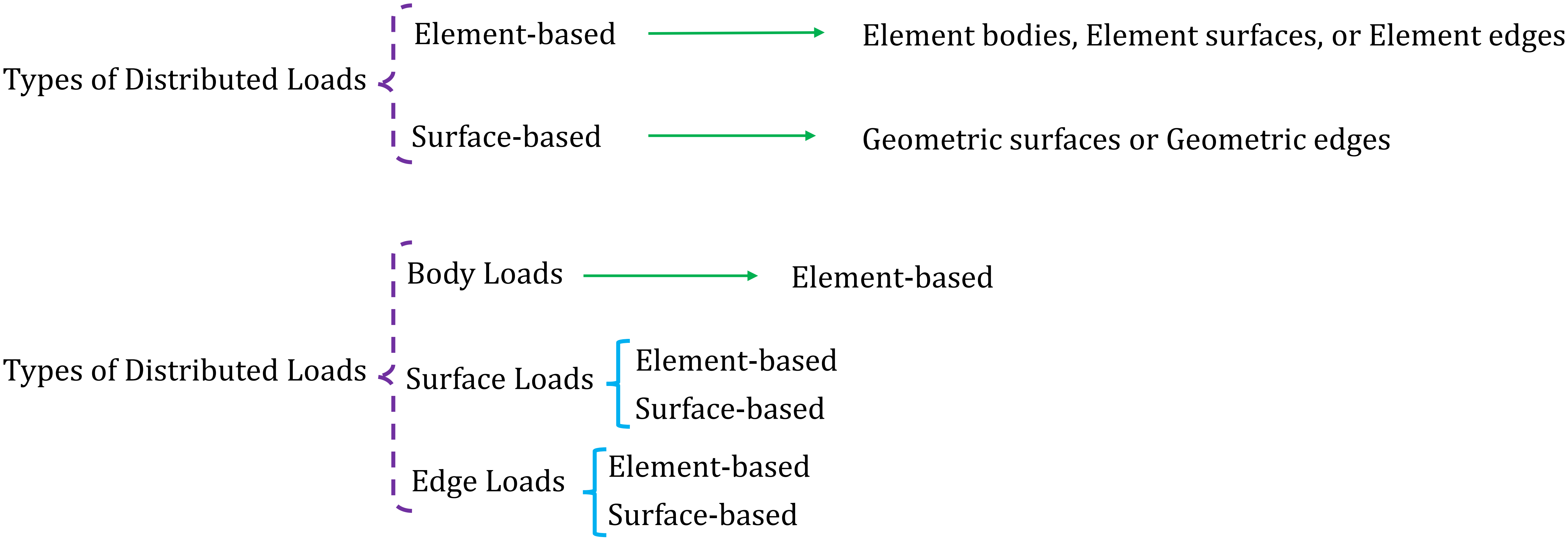
The load type
- Surface-based Load → j in the load type identifies the face or edge of the element underlying the surface
- Edge-based Load → Edge Number
- Element-based Surface Load → Face Number

SNAME → Surface name for a **surface-based** load definition. For an element-based or edge-based load the surface name is passed in as blank

SNAME (Surface Name)
- Part → ASSEMBLY_PART-#_SURFACENAME
- Assembly → ASSEMBLY_SURFACENAME

UTRACLOAD

Distributed Loads Can Be Defined As Element-based Or Surface-based



Types of Distributed Loads

- Element-based → Element bodies, Element surfaces, or Element edges
- Surface-based → Geometric surfaces or Geometric edges

Types of Distributed Loads

- Body Loads → Element-based
- Surface Loads
  - Element-based
  - Surface-based
- Edge Loads
  - Element-based
  - Surface-based

UTRACLOAD

Load Description	Load Label	JLTYP
Nonuniform shear surface traction	TRSHRNU	510+j
	TRSHR1NU	511
	TRSHR2NU	512
	TRSHR3NU	513
	TRSHR4NU	514
	TRSHR5NU	515
	TRSHR6NU	516
Nonuniform general surface traction	TRVECNU	520+j
	TRVEC1NU	521
	TRVEC2NU	522
	TRVEC3NU	523
	TRVEC4NU	524
	TRVEC5NU	525
	TRVEC6NU	526

Load Description	Load Label	JLTYP
Nonuniform general edge traction	EDLDNU	540+j
	EDLD1NU	543
	EDLD2NU	544
	EDLD3NU	545
	EDLD4NU	546
Nonuniform normal edge traction	EDNORNU	550+j
	EDNOR1NU	553
	EDNOR2NU	554
	EDNOR3NU	555
	EDNOR4NU	556
Nonuniform shear edge traction	EDSHRNU	560+j
	EDSHRNU	563
	EDSHRNU	564
	EDSHRNU	565
	EDSHRNU	566

Load Description	Load Label	JLTYP
Nonuniform transverse edge traction	EDTRANU	570+j
	EDTRANU	573
	EDTRANU	574
	EDTRANU	575
	EDTRANU	576
Nonuniform edge moment	EDMOMNU	580+j
	EDMOM1NU	583
	EDMOM2NU	584
	EDMOM3NU	585
	EDMOM4NU	586

UTRACLOAD

COORDS

JLTYP

NOEL

SNAME

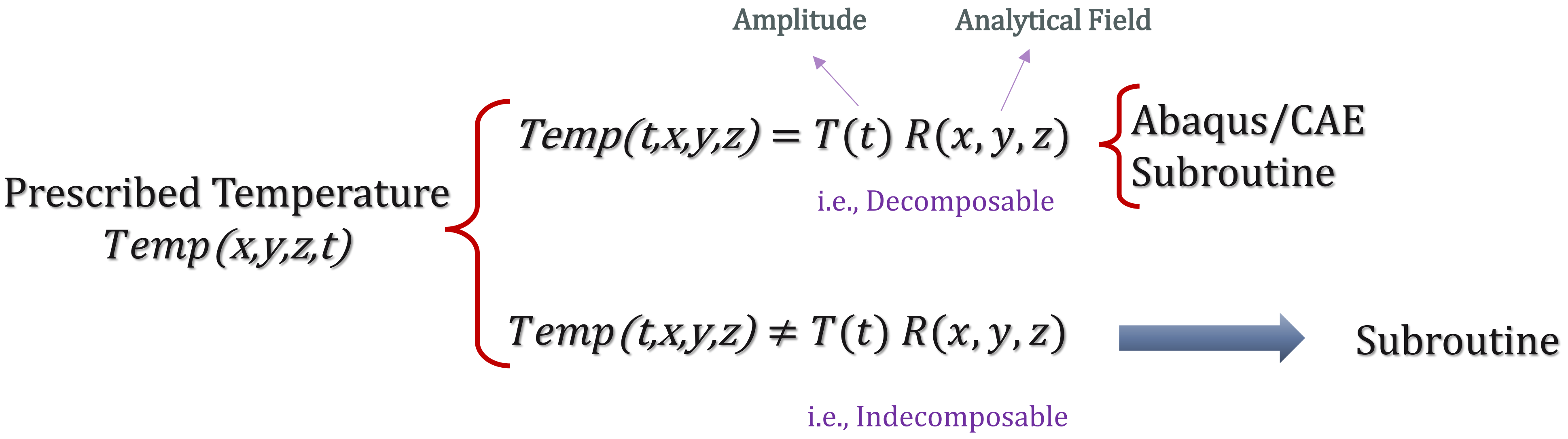
NPT

DIRCOS

UTEMP

Abaqus User Subroutine To Specify Prescribed Temperature

Note the close similarity between the UTEMP and DISP Subroutines



Prescribed Temperature

$Temp(x,y,z,t)$

Amplitude

Analytical Field

$$Temp(t,x,y,z) = \overset{\text{Amplitude}}{T(t)}\ \overset{\text{Analytical Field}}{R(x,y,z)}$$

i.e., Decomposable

Abaqus/CAE

Subroutine

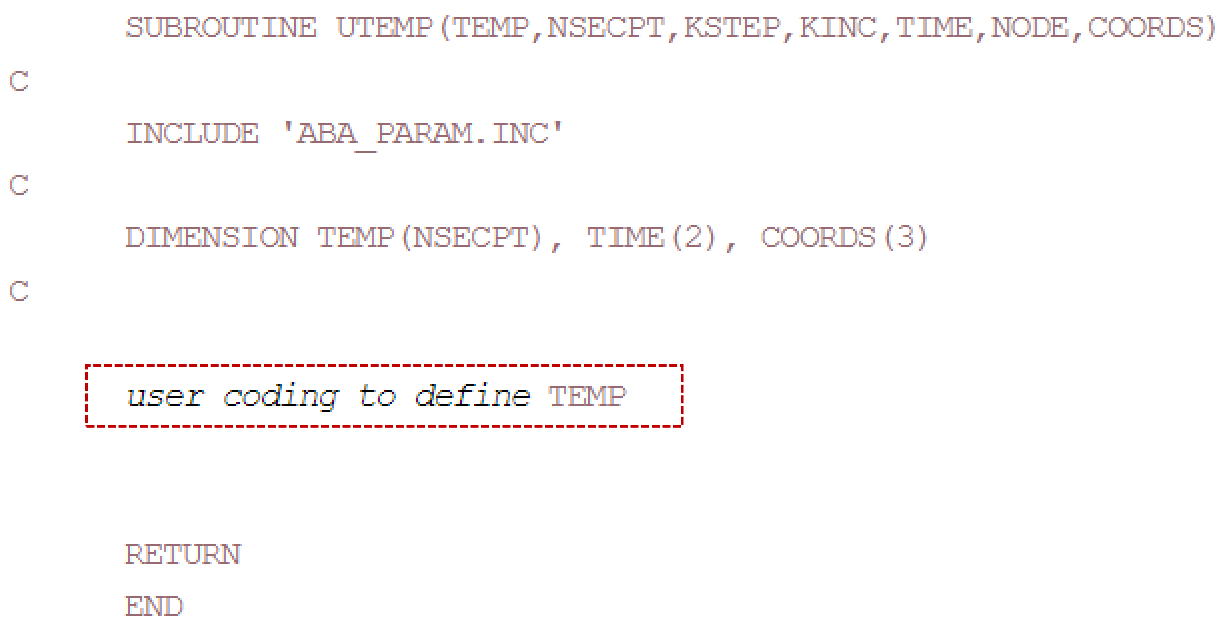
$$Temp(t,x,y,z) \neq T(t)\ R(x,y,z)$$

i.e., Indecomposable

Subroutine

UTEMP

Abaqus User Subroutine To Specify Prescribed Temperature



```
      SUBROUTINE UTEMP(TEMP,NSECPT,KSTEP,KINC,TIME,NODE,COORDS)
C
      INCLUDE 'ABA_PARAM.INC'
C
      DIMENSION TEMP(NSECPT), TIME(2), COORDS(3)
C

      user coding to define TEMP


      RETURN
      END
```

TEMP		Array of temperature values at node number NODE
NSECPT		Maximum number of section values required for any node in the model
KSTEP		Step number
KINC		Increment number
TIME	TIME(1)	Current value of step time
	TIME(2)	Current value of total time
NODE		Node number
COORDS		An array containing the current coordinates of this point.

UTEMP

TEMP		Array of temperature values at node number NODE
NSECPT		Maximum number of section values required for any node in the model
KSTEP		Step number
KINC		Increment number
TIME	TIME(1)	Current value of step time
	TIME(2)	Current value of total time
NODE		Node number
COORDS		An array containing the current coordinates of this point.

- If the node is not connected to a **beam** or **shell** element ⇨ NSECPT=1
- Otherwise
  - Beam Section
    - NSECPT, is determined by the particular section type
    - Origin of the cross-section together with gradients
      - 2D NSECPT=2
      - 3D NSECPT=3
  - Shell Section
    - *n* equally spaced points through each layer ⇨ NSECPT=*n*
    - Reference surface together with gradients with respect to the thickness ⇨ NSECPT=2

FILM

Abaqus User Subroutine To Define Non-uniform Film Coefficient and Associated Sink Temp for Heat Transfer Analysis

$$\mathbf{q}'' = h\,(T_s - T_\infty)$$

Convective Heat Flux

Convection Heat Transfer Coefficient $\left(\frac{W}{K}\right)$ ⇒ Film Coefficient

Fluid Temperature ⇒ Sink Temperature

Surface Temperature

Film Coefficient and Associated Sink Temp

Amplitude

Analytical Field

$$h(t,x,y,z) = T(t)\,R(x,y,z)$$

$$\theta_0(t,x,y,z) = T(t)\,R(x,y,z)$$

i.e., Decomposable ⇒ Abaqus/CAE, Subroutine

i.e., Indecomposable

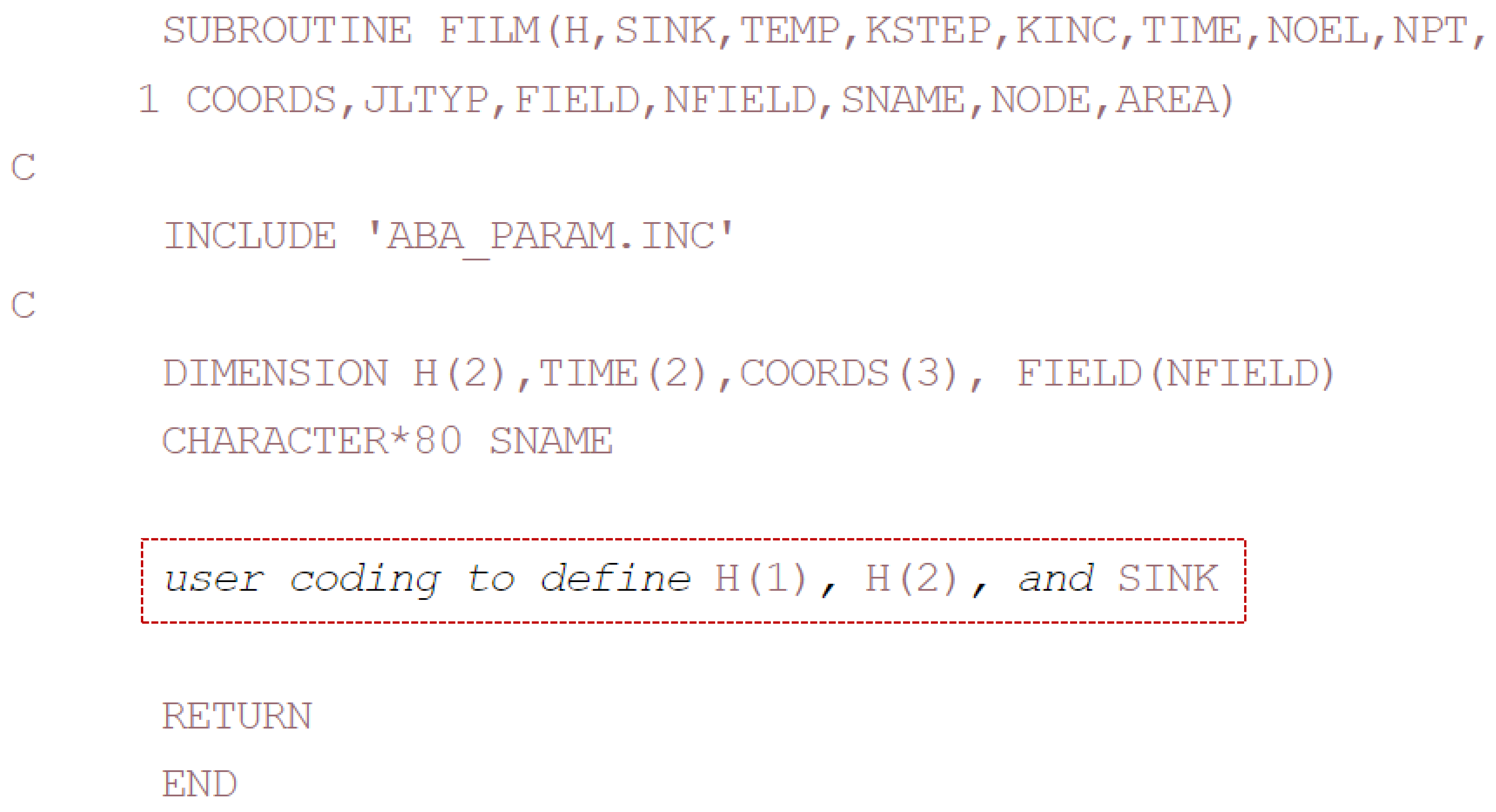
$$h(t,x,y,z) \neq T(t)\,R(x,y,z)$$

$$\theta_0(t,x,y,z) \neq T(t)\,R(x,y,z)$$

⇒ Subroutine

FILM

Abaqus User Subroutine To Define Non-uniform Film Coefficient and Associated Sink Temp for Heat Transfer Analysis



```
      SUBROUTINE FILM(H,SINK,TEMP,KSTEP,KINC,TIME,NOEL,NPT,
     1 COORDS,JLTYP,FIELD,NFIELD,SNAME,NODE,AREA)
C
      INCLUDE 'ABA_PARAM.INC'
C
      DIMENSION H(2),TIME(2),COORDS(3), FIELD(NFIELD)
      CHARACTER*80 SNAME

      user coding to define H(1), H(2), and SINK

      RETURN
      END
```

Heat Transfer

Conduction

Fourier's law →

Temperature Gradient

$$\mathbf{q}'' = -k\nabla T = -k\left(\frac{\partial T}{\partial x}\boldsymbol{i} + \frac{\partial T}{\partial y}\boldsymbol{j} + \frac{\partial T}{\partial z}\boldsymbol{k}\right)$$

Rate of heat energy transfer per unit surface area normal to the direction of transport

(W/m^2)

The direction of Heat Flux is normal to the cross-sectional area

Thermal Conductivity $\left(\frac{W}{m\ K}\right)$

Convention

Newton's law →

Fluid Temperature → Sink Temperature

$$\mathbf{q}'' = h(T_s - T_\infty)$$

Surface Temperature

Convective Heat Flux

Convection Heat Transfer Coefficient $\left(\frac{W}{m^2\ K}\right)$ → Film Coefficient

Radiation

Absolute Temperature (K) Of The Surface

$$q'' = \varepsilon\sigma(T_s^4 - T_{sur}^4)$$

Radiation Heat Flux

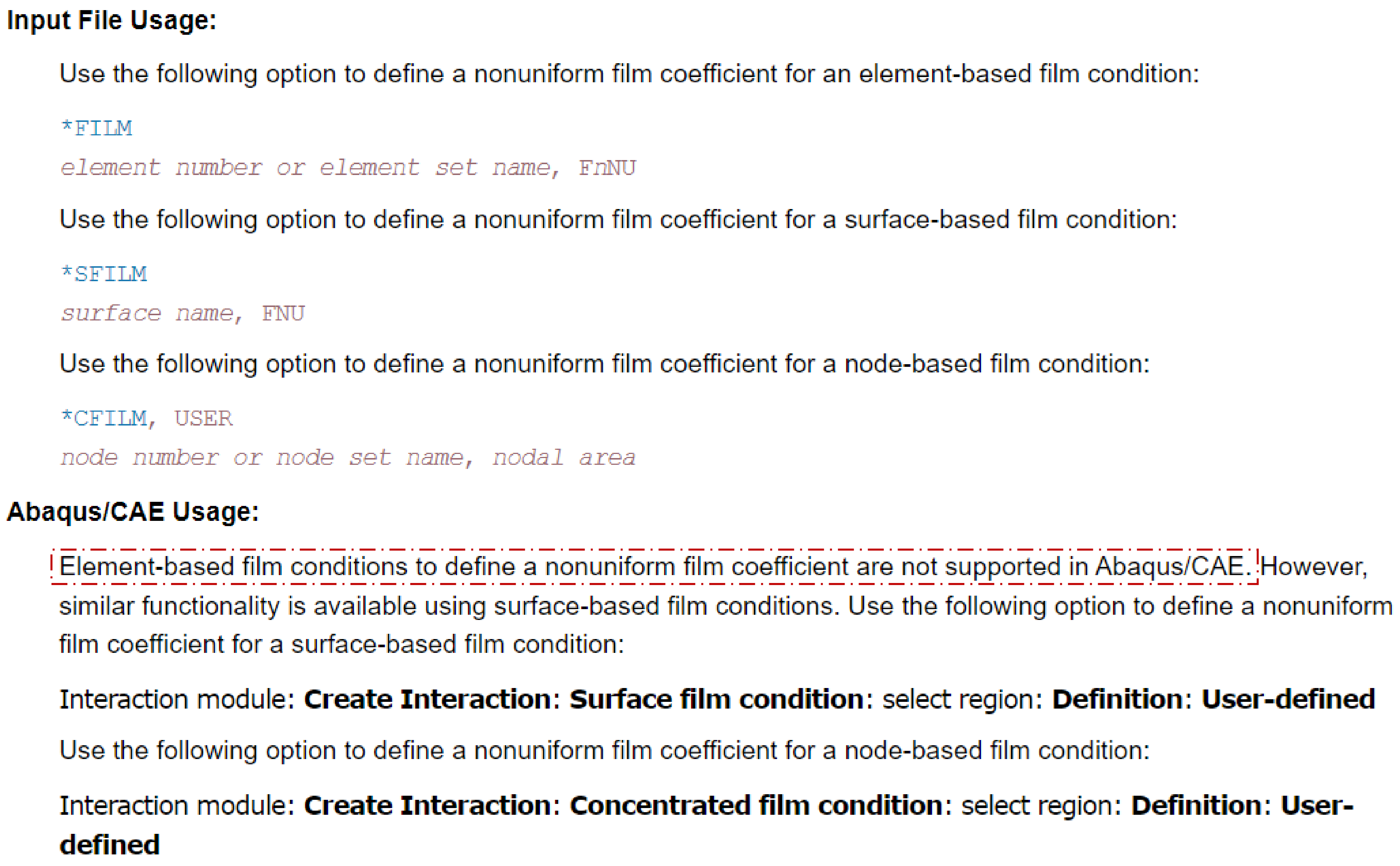
Absolute Temperature (K) Surrounding

emissivity

Stefan–Boltzmann constant

$5.67 \times 10^{-8}\ \left(\frac{W}{m^2K^4}\right)$

Film Coefficient $\left(\frac{J}{TL^2\theta}\right)$: Node-based / Surface-based / Element-based



Input File Usage:

Use the following option to define a nonuniform film coefficient for an element-based film condition:

```
*FILM
element number or element set name, FnNU
```

Use the following option to define a nonuniform film coefficient for a surface-based film condition:

```
*SFILM
surface name, FNU
```

Use the following option to define a nonuniform film coefficient for a node-based film condition:

```
*CFILM, USER
node number or node set name, nodal area
```

Abaqus/CAE Usage:

Element-based film conditions to define a nonuniform film coefficient are not supported in Abaqus/CAE. However, similar functionality is available using surface-based film conditions. Use the following option to define a nonuniform film coefficient for a surface-based film condition:

Interaction module: **Create Interaction**: **Surface film condition**: select region: **Definition**: **User-defined**

Use the following option to define a nonuniform film coefficient for a node-based film condition:

Interaction module: **Create Interaction**: **Concentrated film condition**: select region: **Definition**: **User-defined**

Variables to Be Defined

Film Coefficient
- Node-based
- Element-based
- Surface-based

Sink Temperature
- Node-based
- Element-based
- Surface-based

H

H(1) Magnitude of the Film coefficient at this point $\left(\frac{J}{TL^2\theta}\right)$

H(2) Rate of change of the film coefficient with respect to the surface temperature at this point $\left(\frac{J}{TL^2\theta^2}\right)$

$dh/d\theta$

The rate of convergence during the solution of the nonlinear equations in an increment is improved by defining this value, especially when the film coefficient is a strong function of surface temperature

SINK Sink Temperature

Variables Passed in for Information

TEMP		Estimated Surface Temperature At This Time At This Point
KSTEP		Step Number
KINC		Increment Number
TIME	TIME(1)	Current value of step time
	TIME(2)	Current value of total time
NOEL		Element number (This variable is passed in as zero for node-based films)
NPT		Surface integration point number (This variable is passed in as zero for node-based films)
COORDS		An array containing the coordinates of this point. These are the current coordinates if geometric nonlinearity is accounted for during the step; otherwise, the array contains the original coordinates of the point.

Variables Passed in for Information

JLTYP

Identifies the element face for which this call to FILM is being made for an element-based film coefficient specification

	JLTYP	Film type
	0	Node-based or surface-based loading
Bottom	11	F1NU (FNEGNU for heat transfer shells)
Top	12	F2NU (FPOSNU for heat transfer shells)
	13	F3NU
	14	F4NU
	15	F5NU
	16	F6NU

FIELD

Interpolated values of field variables at this point

NFIELD

Number of field variables

SNAME

Surface name for which this call to FILM is being made for a **surface-based film coefficient** specification (JLTYP=0).
(This variable is passed in as blank for both node-based and element-based films)

NODE

Node Number
(This variable is passed in as zero for both element-based and surface-based films)

AREA

Nodal area for node-based films. AREA will be passed into the routine as the nodal area specified as part of the node-based film coefficient specification.
(This variable is passed in as zero for both element-based and surface-based films)

Example

Transient Heat Transfer

	$SI\ (m)$
Density	$\rho = 7800$
Thermal Conductivity	$k = 1.4$
Specific Heat	$c_p = 260$
Film Coefficient	$h = 10 + 0.2\theta$
Sink Temperature	$\theta_0 = 100 + 2t$
Initial Temperature	$\theta_i = 30$

DFLUX

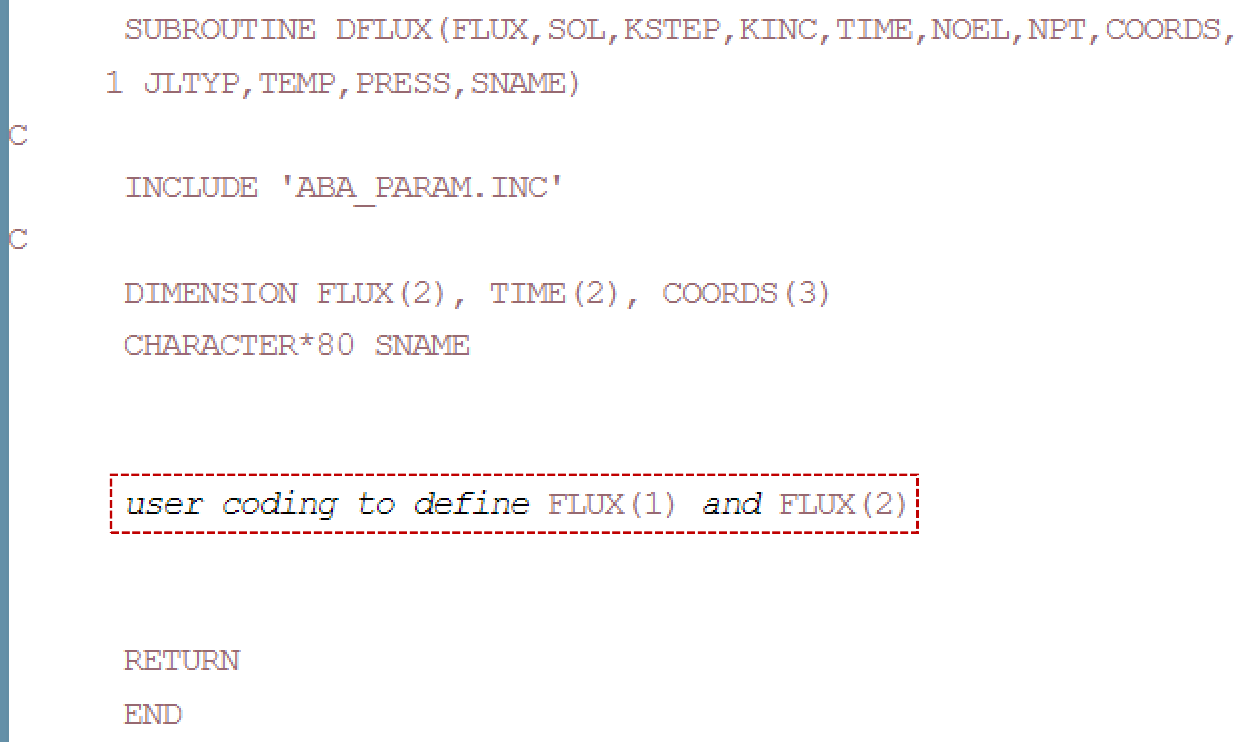
Abaqus User Subroutine To Define Non-uniform Distributed Flux in a Heat Transfer or Mass Diffusion Analysis

Note the close similarity between the DFLUX and DLOAD Subroutines

$$\text{Distributed Flux } q(x,y,z,T,t) \begin{cases} q(x,y,z,t) = \overset{\text{Amplitude}}{T(t)}\;\overset{\text{Analytical Field}}{R(x,y,z)} \quad \left\{\begin{array}{l}\text{Abaqus/CAE}\\ \text{Subroutine}\end{array}\right. \\ \qquad \text{i.e., Decomposable} \\ q(t,x,y,z) \neq T(t)\;R(x,y,z) \quad \xrightarrow{\qquad} \quad \text{Subroutine} \\ \qquad \text{i.e., Indecomposable} \end{cases}$$

DFLUX

Abaqus User Subroutine To Define Non-uniform Distributed Flux in a Heat Transfer or Mass Diffusion Analysis



```
      SUBROUTINE DFLUX(FLUX,SOL,KSTEP,KINC,TIME,NOEL,NPT,COORDS,
     1 JLTYP,TEMP,PRESS,SNAME)
C
      INCLUDE 'ABA_PARAM.INC'
C
      DIMENSION FLUX(2), TIME(2), COORDS(3)
      CHARACTER*80 SNAME


      user coding to define FLUX(1) and FLUX(2)


      RETURN
      END
```

FLUX	FLUX(1)	Magnitude of flux	
	FLUX(2)	Rate of change of the flux with respect to the temperature/mass concentration	
SOL		Estimated value of the solution variable	
KSTEP		Step number	
KINC		Increment number	
TIME	TIME(1)	Current value of step time	Only in transient analysis
	TIME(2)	Current value of total time	
NOEL		Element number	
NPT		Integration point number	
COORDS		An array containing the coordinates of this point (NODE)	
JLTYP		Identifies the flux type	Only for a mass diffusion analysis
TEMP		Current value of temperature at this integration point	

PRESS	Current value of the equivalent pressure stress at this integration point	Only for a mass diffusion analysis
SNAME	Surface name for a surface-based flux definition (JLTYP=0).	

- FLUX
  - Heat Flux: ⇒ Rate of heat energy transfer per unit surface area normal to the direction of transport + Volume
  - Mass Diffusion Flux: ⇒ Rate of mass transfer per unit surface area normal to the direction of transport + Volume
- FLUX
  - Element-based ⇒ Body Flux / Surface Flux
  - Surface-based ⇒ Surface Flux
- FLUX
  - FLUX(1)
    - In transient heat transfer cases where a **nondefault** amplitude is used to vary the applied fluxes, the **time average flux over** the time increment must be defined rather than the value at the end of the time increment
    - Magnitude of flux
      - Surface Flux: $\left(\frac{J}{TL^2}\right) / \left(\frac{PL}{T}\right)$
      - Body Flux: $\left(\frac{J}{TL^3}\right) / \left(\frac{P}{T}\right)$
  - FLUX(2)
    - Rate of change of the flux with respect to the temperature
    - Rate of change of the flux with respect to the mass concentration
    - Heat Transfer $dq/d\theta$
      - Surface Flux: $\left(\frac{J}{TL^2\theta}\right)$
      - Body Flux: $\left(\frac{J}{TL^3\theta}\right)$
    - Mass Diffusion dq/dc
      - Surface Flux: $\left(\frac{L}{T}\right)$
      - Body Flux: $\left(\frac{1}{T}\right)$
    - The **convergence rate** during the solution of the **nonlinear** equations in an increment is improved by defining this value, especially when the flux is a strong function of temperature in heat transfer analysis or concentration in mass diffusion analysis

Abaqus Conventions

Dimension	Indicator	Example (S.I. units)
Length	L	Meter
Mass	M	Kilogram
Time	T	Second
Temperature	θ	Degree Celsius
Electric Current	A	Ampere
Force	F	Newton
Energy	J	Joule
Electric Charge	C	Coulomb
Electric Potential	φ	Volt
Mass Concentration	P	Parts Per Million
Fluid Electric Potential	φ_e	Volt
Ion Concentration In The Electrolyte	C_e	Mol Per Cubic Meter

SOL

Estimated value of the solution variable

temperature in a heat transfer analysis
or
concentration in a mass diffusion analysis

COORDS

An array containing the coordinates of this point (NODE). These are the current coordinates if geometric nonlinearity is accounted for during the step; otherwise, the array contains the original coordinates of the point.

JLTYP

Identifies the flux type → NEXT SLIDE

TEMP

Current value of temperature at this integration point → Only For A Mass Diffusion Analysis

PRESS

Current value of the equivalent pressure stress at this integration point → Only For A Mass Diffusion Analysis

SNAME

Surface name for a surface-based flux definition (JLTYP=0). For a body flux or an element-based surface flux the surface name is passed in as blank.

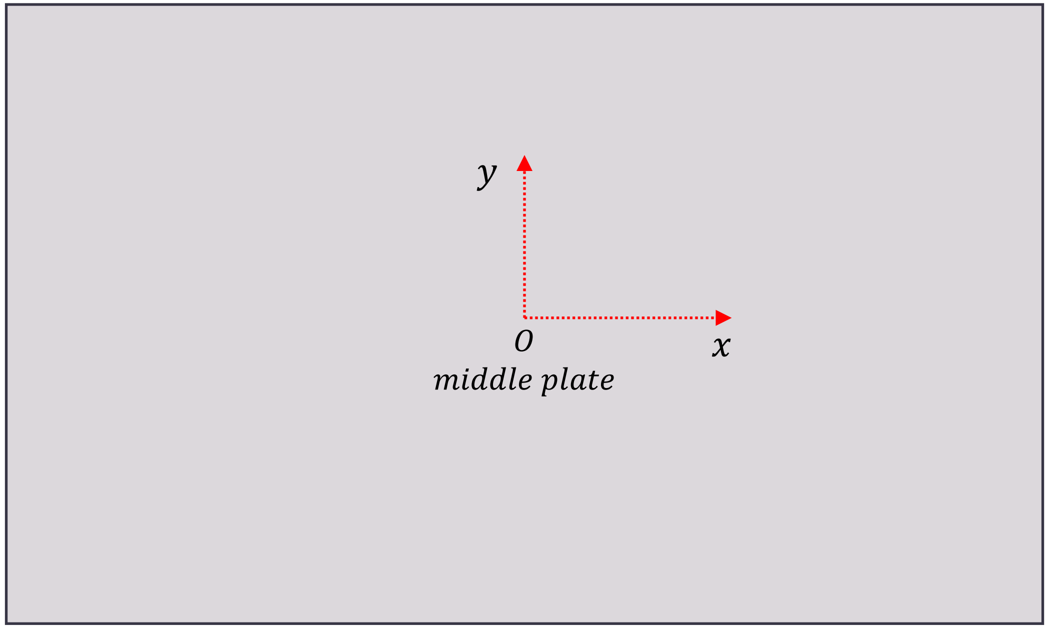
Flux Identifier

JLTYP	Flux Type	Description
0	Surface-based flux	Nonuniform Surface Flux
1	BFNU	Nonuniform **body flux** per unit volume with magnitude supplied via user subroutine DFLUX
11	S1NU (SNEGNU for heat transfer shells)	Nonuniform **surface flux** per unit area into the bottom face of the element with magnitude supplied via user subroutine DFLUX
12	S2NU (SPOSNU for heat transfer shells)	Nonuniform **surface flux** per unit area into the top face of the element with magnitude supplied via user subroutine DFLUX.
13	S3NU	Nonuniform **surface flux** per unit area into the face 3 of the element
14	S4NU	Nonuniform **surface flux** per unit area into the face 4 of the element
15	S5NU	Nonuniform **surface flux** per unit area into the face 5 of the element
16	S6NU	Nonuniform **surface flux** per unit area into the face 6 of the element

Example

$$q\,(x, y, z, t) = cos(10\pi t)\ sin\left(\frac{\pi x}{100}\right) sin\left(\frac{\pi y}{50}\right)$$

$$q\,(x, y, z, \theta, t) = e^{\theta} + cos(10\pi t)\ sin\left(\frac{\pi x}{100}\right) sin\left(\frac{\pi y}{50}\right)$$



$$200\ mm \times 100\ mm \times 1\ mm$$

Material Constant

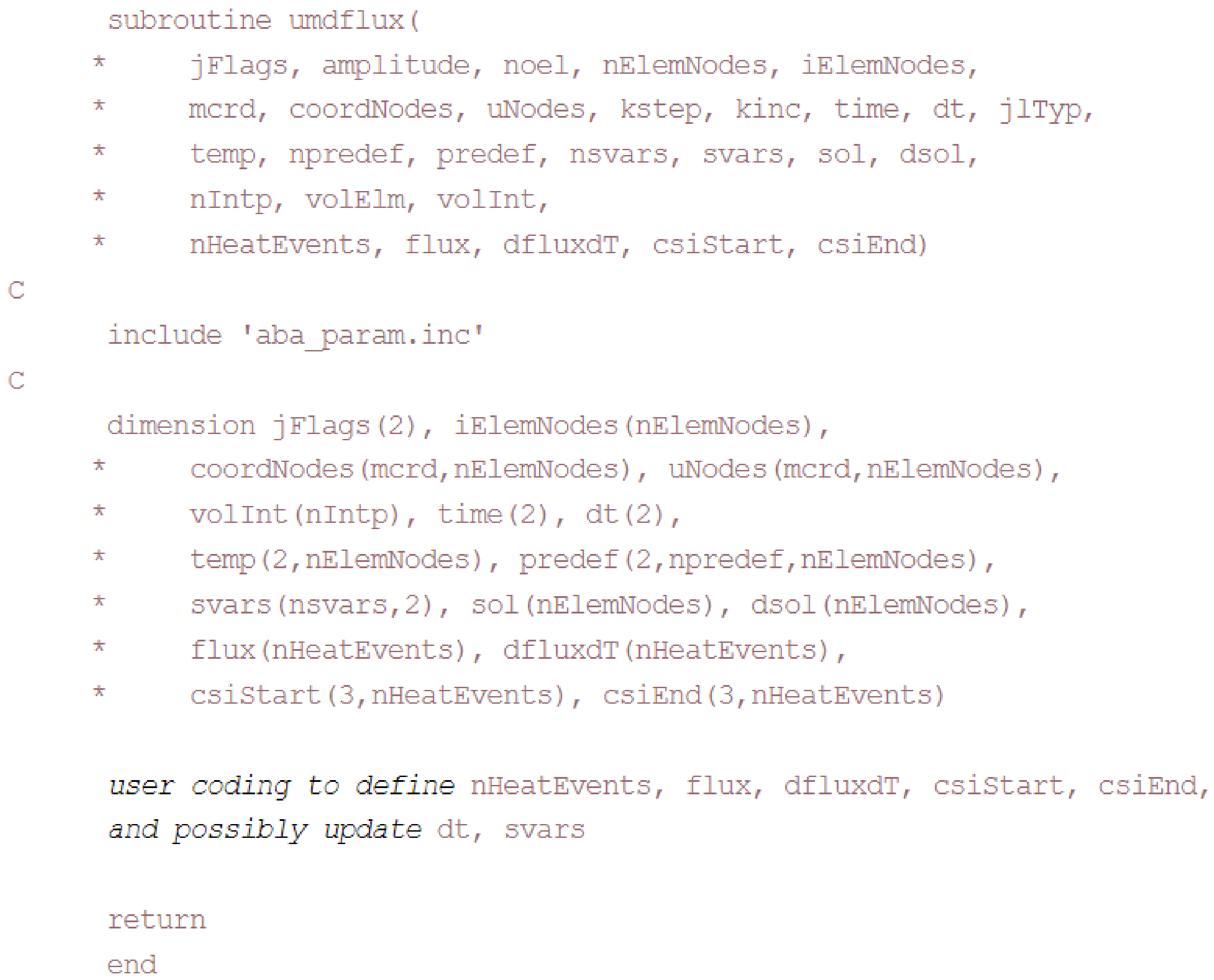
	Commonly used unit	SI value	SI (mm) value
Stiffness of steel	$210\ GPa$	210×10^9 Pa	$210000\ MPa$
Density of steel	$7850\ \frac{kg}{m^3}$		$7.85 \times 10^{-9}\ \frac{tonne}{mm^3}$
Gravitational constant	$9.81\ \frac{m}{s^2}$		$9810\frac{mm}{s^2}$
pressure	$1\ bar$	105 Pa	$0.1\ MPa$
Absolute zero temperature	-273.15 °C	0 K	°C and K both acceptable
Stefan-Boltzmann constant	$5.67 \times 10^{-8}\ \frac{W}{m^2K^4}$		$5.67 \times 10^{-11}\ \frac{mW}{mm^2K^4}$
Universal gas constant	$8.31446\ \frac{J}{K\ mol}$		$8314.46\ \frac{mJ}{K\ mol}$

Conductivity $45\ \frac{W}{m\ K} = \frac{m\ W}{mm\ K}$

Specific Heat $420\ \frac{J}{kg\ k} = 420 \times 10^6\ \frac{mJ}{ton\ K}$

UMDFLUX

Abaqus User Subroutine To Specifying Moving or Stationary Nonuniform Heat Flux in a Heat Transfer Analysis



```
      subroutine umdflux(
     *     jFlags, amplitude, noel, nElemNodes, iElemNodes,
     *     mcrd, coordNodes, uNodes, kstep, kinc, time, dt, jlTyp,
     *     temp, npredef, predef, nsvars, svars, sol, dsol,
     *     nIntp, volElm, volInt,
     *     nHeatEvents, flux, dfluxdT, csiStart, csiEnd)
C
      include 'aba_param.inc'
C
      dimension jFlags(2), iElemNodes(nElemNodes),
     *     coordNodes(mcrd,nElemNodes), uNodes(mcrd,nElemNodes),
     *     volInt(nIntp), time(2), dt(2),
     *     temp(2,nElemNodes), predef(2,npredef,nElemNodes),
     *     svars(nsvars,2), sol(nElemNodes), dsol(nElemNodes),
     *     flux(nHeatEvents), dfluxdT(nHeatEvents),
     *     csiStart(3,nHeatEvents), csiEnd(3,nHeatEvents)

      user coding to define nHeatEvents, flux, dfluxdT, csiStart, csiEnd,
      and possibly update dt, svars

      return
      end
```

JLTYP

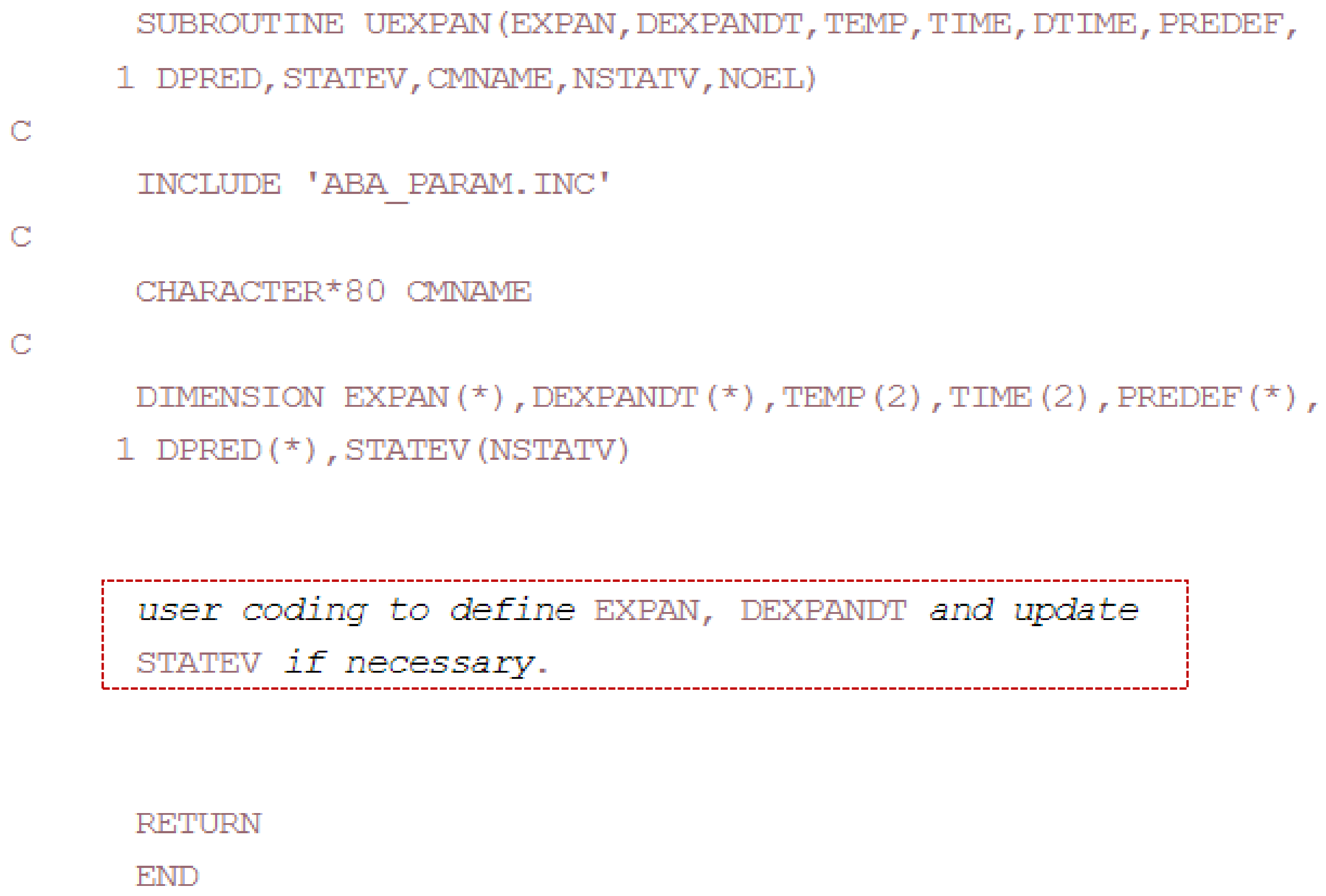
Identifies the moving flux type for which this call to UMDFLUX is being made; only the concentrated heat flux type is supported (JLTYP=1)

JLTYP	Flux Type	Description
0	MBFNU	Nonuniform moving or stationary concentrated heat fluxes with magnitudes supplied via user subroutine UMDFLUX.

UEXPAN

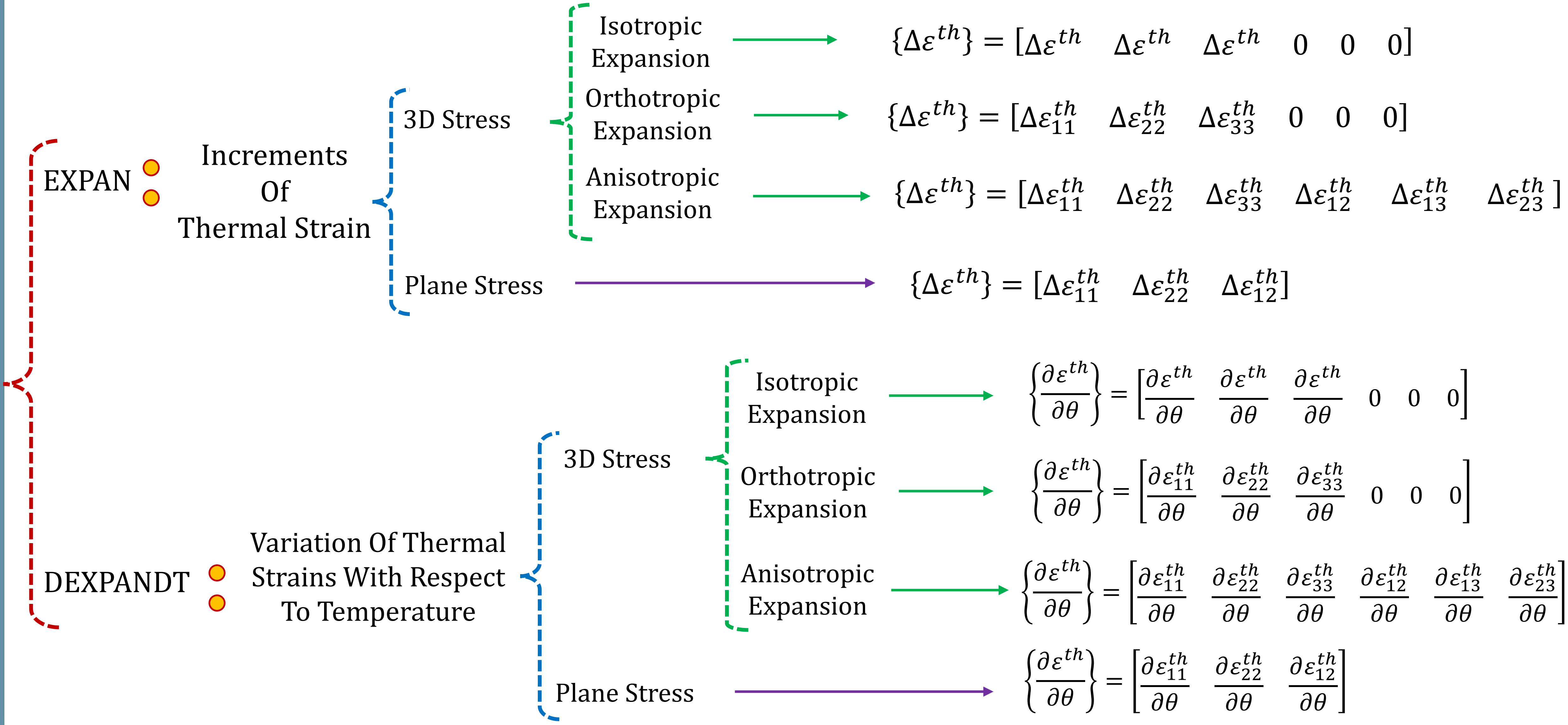
Abaqus User Subroutines To Define Incremental Thermal Strains

Thermal Strains Are Complicated Functions Of Temperature, Time, Field Variables, And State Variables



```
      SUBROUTINE UEXPAN(EXPAN,DEXPANDT,TEMP,TIME,DTIME,PREDEF,
     1 DPRED,STATEV,CMNAME,NSTATV,NOEL)
C
      INCLUDE 'ABA_PARAM.INC'
C
      CHARACTER*80 CMNAME
C
      DIMENSION EXPAN(*),DEXPANDT(*),TEMP(2),TIME(2),PREDEF(*),
     1 DPRED(*),STATEV(NSTATV)


      user coding to define EXPAN, DEXPANDT and update
      STATEV if necessary.


      RETURN
      END
```

Variables to Be Defined



- EXPAN ● ● Increments Of Thermal Strain
  - 3D Stress
    - Isotropic Expansion → $\{\Delta\varepsilon^{th}\} = [\Delta\varepsilon^{th} \quad \Delta\varepsilon^{th} \quad \Delta\varepsilon^{th} \quad 0 \quad 0 \quad 0]$
    - Orthotropic Expansion → $\{\Delta\varepsilon^{th}\} = [\Delta\varepsilon_{11}^{th} \quad \Delta\varepsilon_{22}^{th} \quad \Delta\varepsilon_{33}^{th} \quad 0 \quad 0 \quad 0]$
    - Anisotropic Expansion → $\{\Delta\varepsilon^{th}\} = [\Delta\varepsilon_{11}^{th} \quad \Delta\varepsilon_{22}^{th} \quad \Delta\varepsilon_{33}^{th} \quad \Delta\varepsilon_{12}^{th} \quad \Delta\varepsilon_{13}^{th} \quad \Delta\varepsilon_{23}^{th}\ ]$
  - Plane Stress → $\{\Delta\varepsilon^{th}\} = [\Delta\varepsilon_{11}^{th} \quad \Delta\varepsilon_{22}^{th} \quad \Delta\varepsilon_{12}^{th}]$
- DEXPANDT ● ● Variation Of Thermal Strains With Respect To Temperature
  - 3D Stress
    - Isotropic Expansion → $\left\{\frac{\partial\varepsilon^{th}}{\partial\theta}\right\} = \left[\frac{\partial\varepsilon^{th}}{\partial\theta} \quad \frac{\partial\varepsilon^{th}}{\partial\theta} \quad \frac{\partial\varepsilon^{th}}{\partial\theta} \quad 0 \quad 0 \quad 0\right]$
    - Orthotropic Expansion → $\left\{\frac{\partial\varepsilon^{th}}{\partial\theta}\right\} = \left[\frac{\partial\varepsilon_{11}^{th}}{\partial\theta} \quad \frac{\partial\varepsilon_{22}^{th}}{\partial\theta} \quad \frac{\partial\varepsilon_{33}^{th}}{\partial\theta} \quad 0 \quad 0 \quad 0\right]$
    - Anisotropic Expansion → $\left\{\frac{\partial\varepsilon^{th}}{\partial\theta}\right\} = \left[\frac{\partial\varepsilon_{11}^{th}}{\partial\theta} \quad \frac{\partial\varepsilon_{22}^{th}}{\partial\theta} \quad \frac{\partial\varepsilon_{33}^{th}}{\partial\theta} \quad \frac{\partial\varepsilon_{12}^{th}}{\partial\theta} \quad \frac{\partial\varepsilon_{13}^{th}}{\partial\theta} \quad \frac{\partial\varepsilon_{23}^{th}}{\partial\theta}\right]$
  - Plane Stress → $\left\{\frac{\partial\varepsilon^{th}}{\partial\theta}\right\} = \left[\frac{\partial\varepsilon_{11}^{th}}{\partial\theta} \quad \frac{\partial\varepsilon_{22}^{th}}{\partial\theta} \quad \frac{\partial\varepsilon_{12}^{th}}{\partial\theta}\right]$

Variables Passed in for Information

TEMP	TEMP(1) →	Current Temperature (at the end of the increment)
	TEMP(2) →	Temperature Increment
TIME	TIME(1) →	Step Time At The End Of The Increment
	TIME(2) →	Total Time At The End Of The Increment
DTIME	→	Time Increment
PREDEF	→	Array Containing The Values Of All The User-specified **Predefined Field Variables** At This Point (initial values at the beginning of the analysis and current values during the analysis)
DPRED	→	Array Of Increments Of Predefined Field Variables
CMNAME	→	User-specified Material Name Or Gasket Behavior Name, Left Justified
NOEL	→	User-defined Element Number

Variables That Can Be Updated

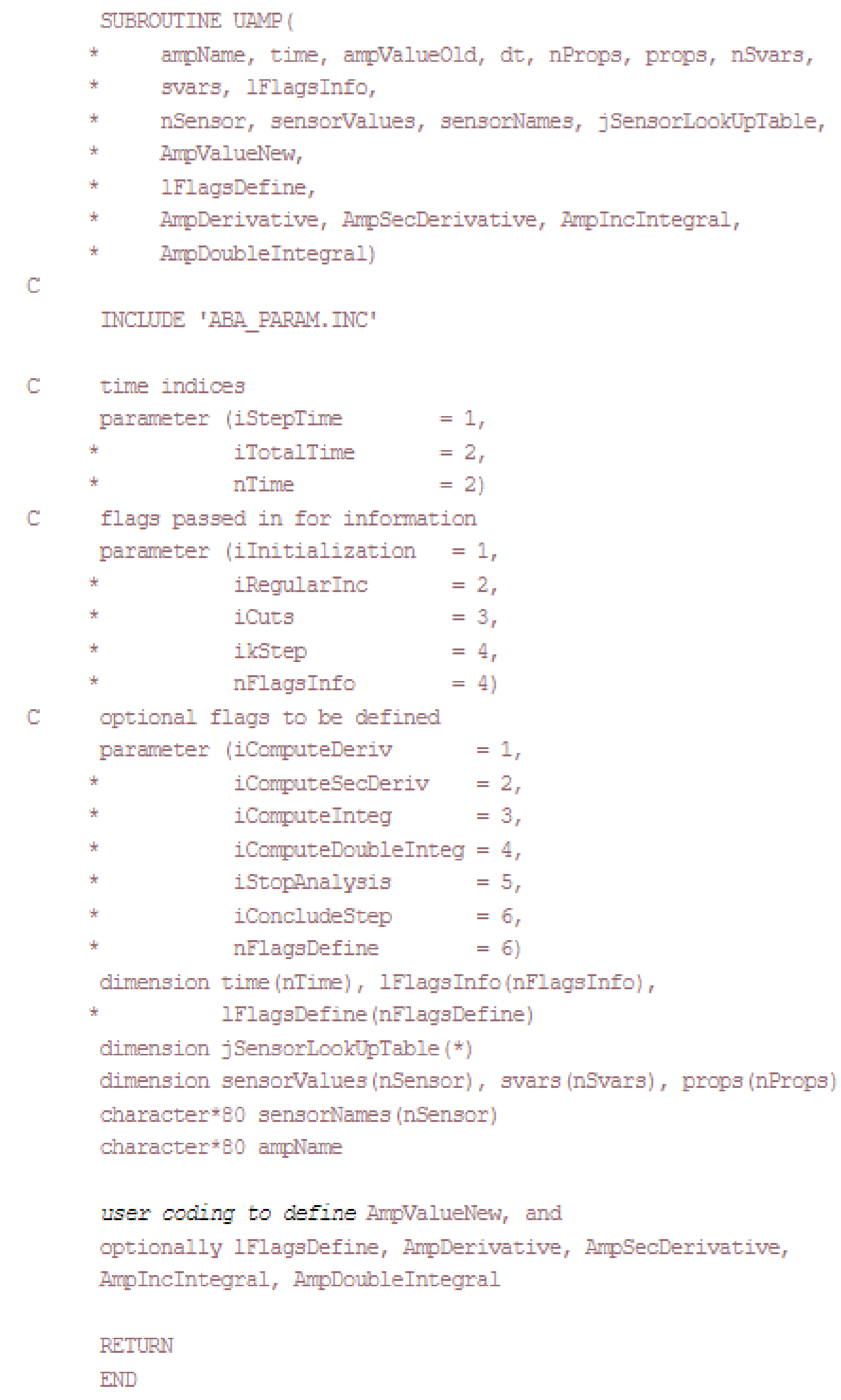
- STATEV
  - Others → Array Containing The User-defined Solution-dependent State Variables At This Point.
  - Coupled Temperature-displacement
    And
    Coupled Thermal-electrical-structural → These are supplied as values at the start of the increment and can be updated to their values at the end of the increment.

    ↓

    UEXPAN is called twice
    Per Material Point Per Iteration.
    - In the first call for a given material point and iteration, the values supplied are those at the start of the increment and can be updated.
    - In the second call for the same material point and iteration, the values supplied are those returned from the first call, and they can be updated again to their values at the end of the increment.

  User subroutine UEXPAN allows for the incremental thermal strains to be **only weakly dependent on the state variables.** The Jacobian terms arising from the derivatives of the thermal strains with respect to the state variables are not taken into account

- NSTATEV → Number of solution-dependent state variables associated with this material or gasket behavior type (specified when space is allocated for the array)



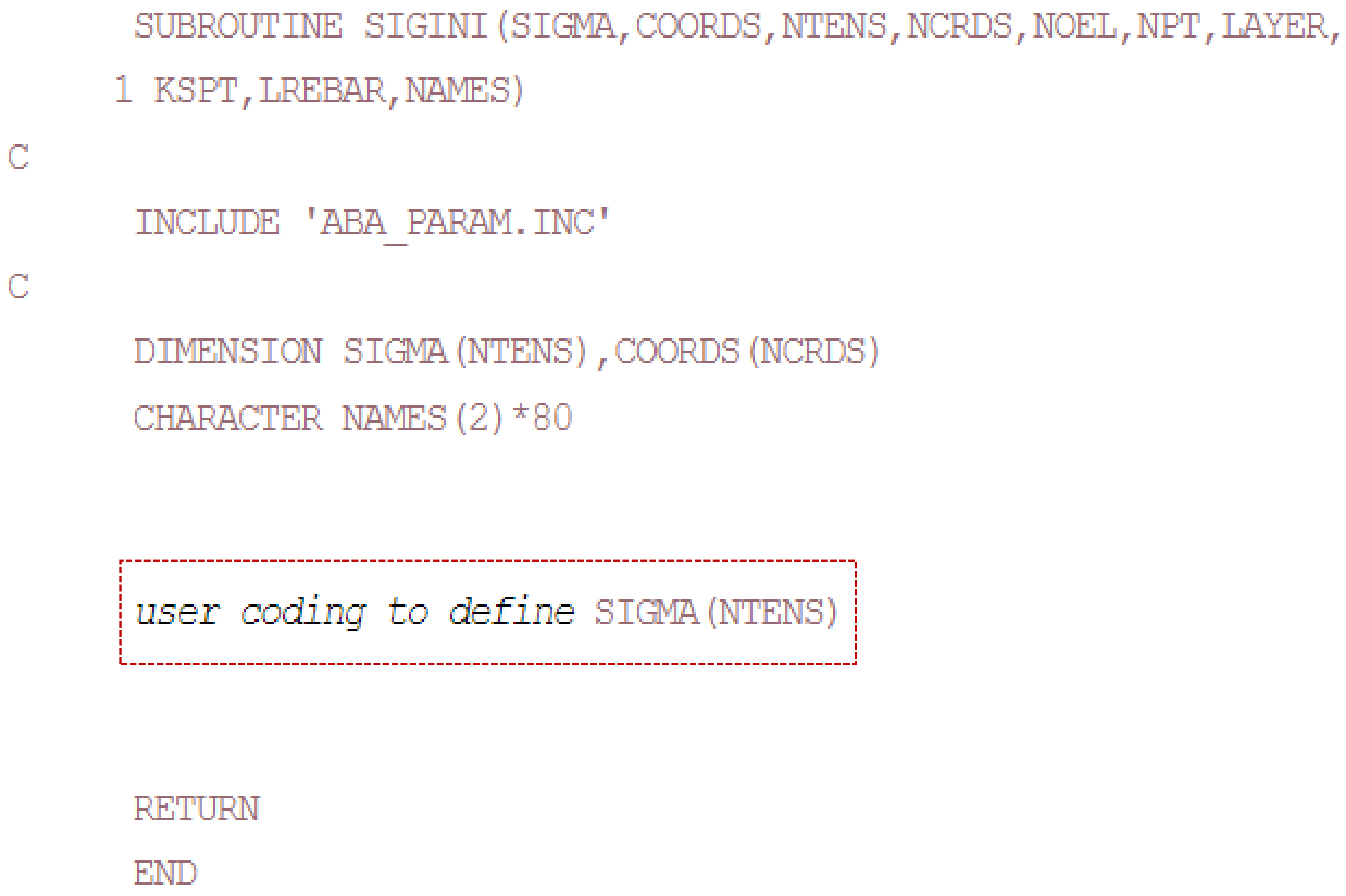
```
      SUBROUTINE UAMP(
     *     ampName, time, ampValueOld, dt, nProps, props, nSvars,
     *     svars, lFlagsInfo,
     *     nSensor, sensorValues, sensorNames, jSensorLookUpTable,
     *     AmpValueNew,
     *     lFlagsDefine,
     *     AmpDerivative, AmpSecDerivative, AmpIncIntegral,
     *     AmpDoubleIntegral)
C
      INCLUDE 'ABA_PARAM.INC'

C     time indices
      parameter (iStepTime        = 1,
     *           iTotalTime       = 2,
     *           nTime            = 2)
C     flags passed in for information
      parameter (iInitialization   = 1,
     *           iRegularInc       = 2,
     *           iCuts             = 3,
     *           ikStep            = 4,
     *           nFlagsInfo        = 4)
C     optional flags to be defined
      parameter (iComputeDeriv       = 1,
     *           iComputeSecDeriv    = 2,
     *           iComputeInteg       = 3,
     *           iComputeDoubleInteg = 4,
     *           iStopAnalysis       = 5,
     *           iConcludeStep       = 6,
     *           nFlagsDefine        = 6)
      dimension time(nTime), lFlagsInfo(nFlagsInfo),
     *          lFlagsDefine(nFlagsDefine)
      dimension jSensorLookUpTable(*)
      dimension sensorValues(nSensor), svars(nSvars), props(nProps)
      character*80 sensorNames(nSensor)
      character*80 ampName

      user coding to define AmpValueNew, and
      optionally lFlagsDefine, AmpDerivative, AmpSecDerivative,
      AmpIncIntegral, AmpDoubleIntegral

      RETURN
      END
```

UAMP

Abaqus User Subroutines To Specify Amplitude

SIGINI

Abaqus User Subroutines To Define An Initial Stress Field



```
      SUBROUTINE SIGINI(SIGMA,COORDS,NTENS,NCRDS,NOEL,NPT,LAYER,
     1 KSPT,LREBAR,NAMES)
C
      INCLUDE 'ABA_PARAM.INC'
C
      DIMENSION SIGMA(NTENS),COORDS(NCRDS)
      CHARACTER NAMES(2)*80


      user coding to define SIGMA(NTENS)


      RETURN
      END
```

Variables to Be Defined

SIGMA(i)	i^{th} stress component
COORDS	An array containing the initial coordinates of this point
NTENS	Number of stresses
NCRDS	Number of coordinates
NOEL	Element number
NPT	Integration point number in the element
LAYER	Layer number
KSTP	Section point number within the current layer
LREBAR	Rebar flag
NAMES	NAMES(1): Name of the rebar NAMES(2): Element type name

Variables Passed in for Information

SIGMA(i)		i^{th} stress component
COORDS		An array containing the initial coordinates of this point
NTENS		Number of stresses
NCRDS		Number of coordinates
NOEL		Element number
NPT		Integration point number in the element
LAYER		Layer number
KSTP		Section point number within the current layer
LREBAR		Rebar flag
NAMES	NAMES(1):	Name of the rebar
	NAMES(2):	Element type name

SIGINI

Abaqus User Subroutines To Define An Initial Stress Field

SIGMA(i)		i^{th} stress component
COORDS		An array containing the current coordinates of this point.
NTENS		Number of stresses
NCRDS		Number of coordinates
NOEL		Element number
NPT		Integration point number in the element
LAYER		Layer number
KSTP		Section point number within the current layer
LREBAR		Rebar flag
NAMES	NAMES(1):	Name of the rebar
	NAMES(2):	Element type name

3D Stress: 6

Axisymmetric, and (Generalized) Plane Strain: 4

Plane Stress: 3

Keyword

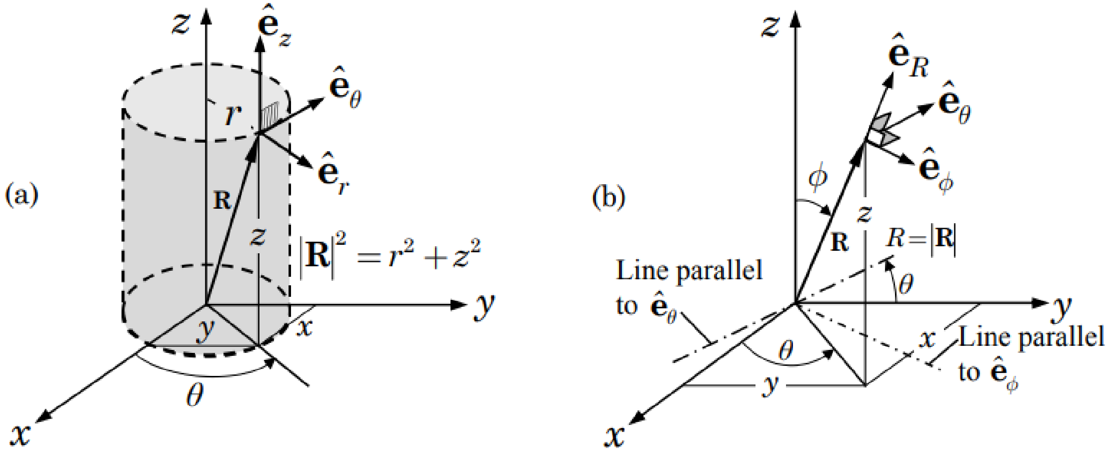
** INITIAL CONDITIONS

* INITIAL CONDITIONS, TYPE=STRESS, USER

SIGINI

$$\begin{Bmatrix} \hat{\mathbf{e}}_R \\ \hat{\mathbf{e}}_\phi \\ \hat{\mathbf{e}}_\theta \end{Bmatrix} = \begin{bmatrix} \sin\phi\cos\theta & \sin\phi\sin\theta & \cos\phi \\ \cos\phi\cos\theta & \cos\phi\sin\theta & -\sin\phi \\ -\sin\theta & \cos\theta & 0 \end{bmatrix} \begin{Bmatrix} \hat{\mathbf{e}}_x \\ \hat{\mathbf{e}}_y \\ \hat{\mathbf{e}}_z \end{Bmatrix}, \qquad (2.4.43)$$

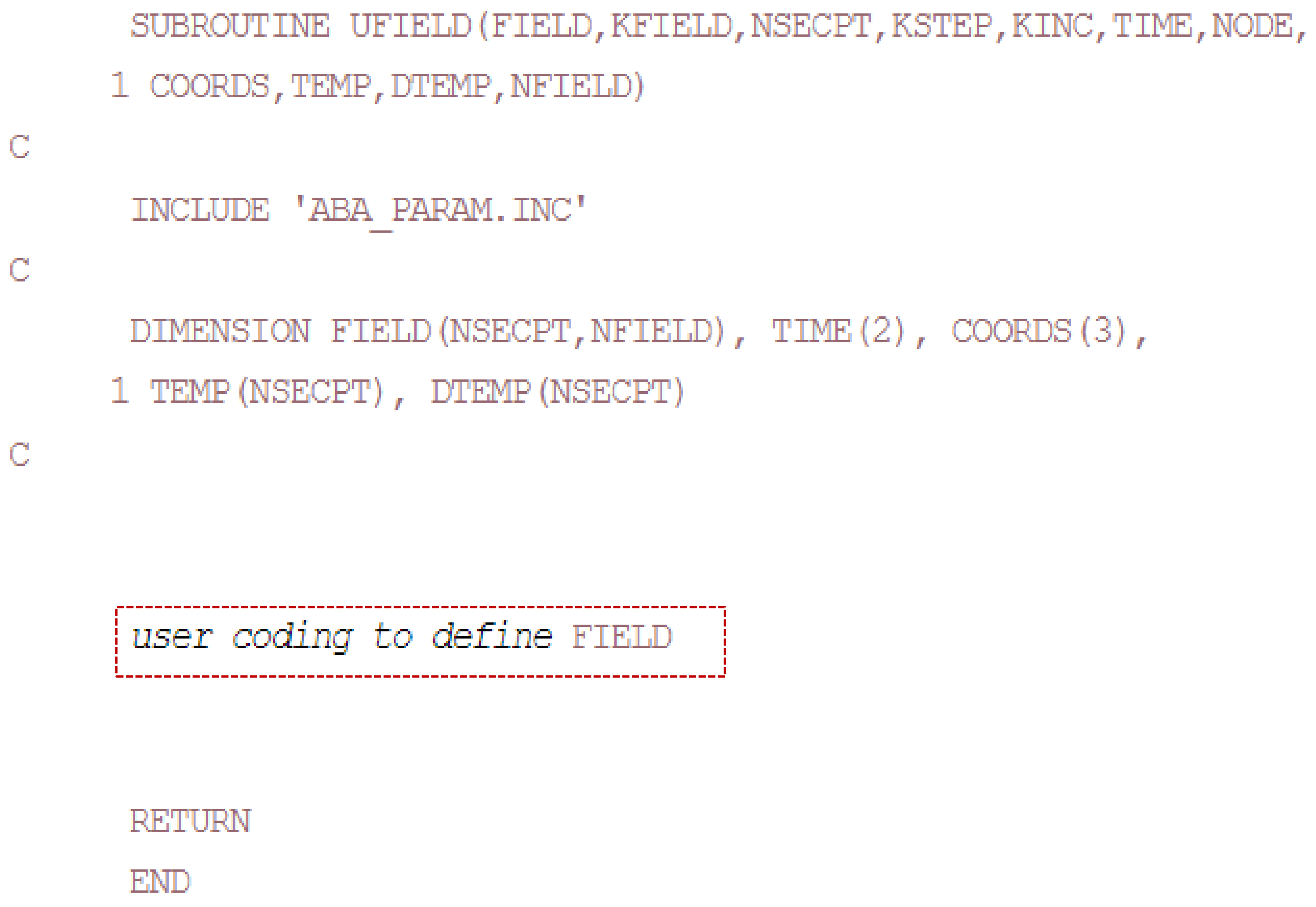
$$\begin{Bmatrix} \hat{\mathbf{e}}_x \\ \hat{\mathbf{e}}_y \\ \hat{\mathbf{e}}_z \end{Bmatrix} = \begin{bmatrix} \sin\phi\cos\theta & \cos\phi\cos\theta & -\sin\theta \\ \sin\phi\sin\theta & \cos\phi\sin\theta & \cos\theta \\ \cos\phi & -\sin\phi & 0 \end{bmatrix} \begin{Bmatrix} \hat{\mathbf{e}}_R \\ \hat{\mathbf{e}}_\phi \\ \hat{\mathbf{e}}_\theta \end{Bmatrix}. \qquad (2.4.44)$$


(a)
z
$\hat{\mathbf{e}}_z$
$\hat{\mathbf{e}}_\theta$
r
$\hat{\mathbf{e}}_r$
R
z
$|\mathbf{R}|^2 = r^2 + z^2$
y
y
x
θ
x
(b)
z
$\hat{\mathbf{e}}_R$
$\hat{\mathbf{e}}_\theta$
$\hat{\mathbf{e}}_\phi$
φ
z
R
$R = |\mathbf{R}|$
θ
Line parallel to $\hat{\mathbf{e}}_\theta$
y
θ
x
Line parallel to $\hat{\mathbf{e}}_\phi$
y
x

$$\bar{s}_{mn} = s_{ij}\,\ell_{mi}\,\ell_{nj} \quad \text{or} \quad [\bar{S}] = [L][S][L]^{\mathrm{T}}.$$

UFIELD

Abaqus User Subroutines To Specify Predefined Field Variables



```
      SUBROUTINE UFIELD(FIELD,KFIELD,NSECPT,KSTEP,KINC,TIME,NODE,
     1 COORDS,TEMP,DTEMP,NFIELD)
C
      INCLUDE 'ABA_PARAM.INC'
C
      DIMENSION FIELD(NSECPT,NFIELD), TIME(2), COORDS(3),
     1 TEMP(NSECPT), DTEMP(NSECPT)
C


      user coding to define FIELD


      RETURN
      END
```

Variables to Be Defined

FIELD(NSECPT, NFIELD) Array Of Predefined Field Variable Values

Array of predefined field variable values at node number NODE. When updating only one field variable at a time, only the value of the specified field variable (see KFIELD below) must be returned. In this case NFIELD is passed into user subroutine UFIELD with a value of 1, and FIELD is thus dimensioned as FIELD(NSECPT,1). When updating all field variables simultaneously, the values of the specified number of field variables at the point must be returned. In this case FIELD is dimensioned as FIELD(NSECPT,NFIELD), where NFIELD is the number of field variables specified and KFIELD has no meaning.

If NODE is part of any element other than a beam or shell, only one value of each field variable must be returned (NSECPT=1). Otherwise, the number of values to be returned depends on the mode of temperature and field variable input selected for the beam or shell section. The following cases are possible:

Temperatures and field variables for a beam section are given as values at the points shown in the beam section descriptions. The number of values required, NSECPT, is determined by the particular section type specified, as described in Beam Cross-Section Library.

Temperatures and field variables are given as values at n equally spaced points through each layer of a shell section. The number of values required, NSECPT, is equal to n.

Temperatures and field variables for a beam section are given as values at the origin of the cross-section together with gradients with respect to the 2-direction and, for three-dimensional beams, the 1-direction of the section; or temperatures and field variables for a shell section are given as values at the reference surface together with gradients through the thickness. The number of values required, NSECPT, is 3 for three-dimensional beams, 2 for two-dimensional beams, and 2 for shells. Give the midsurface value first, followed by the first and (if necessary) second gradients, as described in Beam Elements and Shell Elements.

Since field variables can also be defined directly, it is important to understand the hierarchy used in situations of conflicting information (see Predefined Fields).

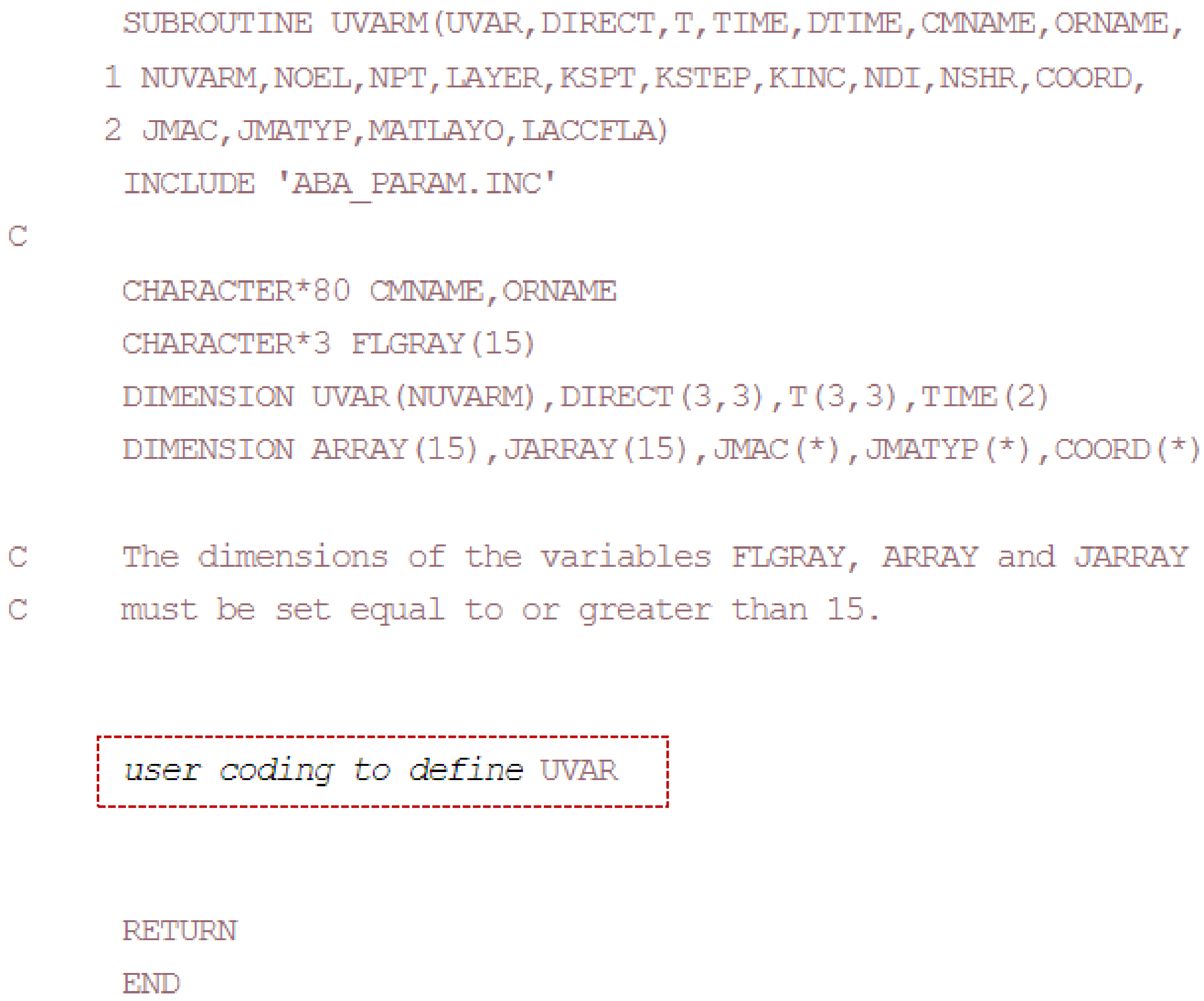
When the array FIELD is passed into user subroutine UFIELD, it will contain either the field variable values from the previous increment or those values obtained from the results file if this method was used. You are then free to modify these values within this subroutine.

Variables Passed in for Information

KFIELD		User-specified field variable number. This variable is meaningful only when updating individual field variables at a time.
NFIELD		User-specified number of field variables to be updated. This variable is meaningful only when updating multiple field variables simultaneously.
NSECPT		Maximum number of section values required for any node in the model
KSTEP		Step Number
KINC		Increment Number
TIME	TIME(1)	Current value of step time
	TIME(2)	Current value of total time
NODE		Node Number
COORDS		An array containing the coordinates of this node. These are the current coordinates if geometric nonlinearity is accounted for during the step; otherwise, the array contains the original coordinates of the node
TEMP(NSECPT)		Current temperature at the node. If user subroutines UTEMP and UFIELD are both used, user subroutine UTEMP is processed before user subroutine UFIELD.
DTEMP(NSECPT)		Temperature increment at the node

UVARM

Abaqus User Subroutines To Generate Element Output



```
      SUBROUTINE UVARM(UVAR,DIRECT,T,TIME,DTIME,CMNAME,ORNAME,
     1 NUVARM,NOEL,NPT,LAYER,KSPT,KSTEP,KINC,NDI,NSHR,COORD,
     2 JMAC,JMATYP,MATLAYO,LACCFLA)
      INCLUDE 'ABA_PARAM.INC'
C
      CHARACTER*80 CMNAME,ORNAME
      CHARACTER*3 FLGRAY(15)
      DIMENSION UVAR(NUVARM),DIRECT(3,3),T(3,3),TIME(2)
      DIMENSION ARRAY(15),JARRAY(15),JMAC(*),JMATYP(*),COORD(*)

C     The dimensions of the variables FLGRAY, ARRAY and JARRAY
C     must be set equal to or greater than 15.


      user coding to define UVAR


      RETURN
      END
```

UVARM

Abaqus User Subroutines To Generate Element Output

UVARM allows you to define output quantities that are functions of any of the available integration point quantities

Will be called at all material calculation points of elements for which the material definition includes the specification of user-defined output variables

Cannot be used with linear perturbation procedures, except for the static perturbation procedure

The data are provided in double precision for output to the data (.dat) and results (.fil) files and are written to the output database (.odb) file in single precision.

Variables to Be Defined

UVAR(NUVARM) ------→ An array containing the user-defined output variables.

These are passed in as the values at the beginning of the increment and must be returned as the values at the end of the increment.

Variables Passed in for Information

DIRECT(3,3) An array containing the direction cosines of the material directions in terms of the global basis directions

First Material Direction →	First Column	DIRECT(1,1), DIRECT(2,1), DIRECT(3,1)
Second Material Direction →	Second Column	DIRECT(1,2), DIRECT(2,2), DIRECT(3,2)
Third Material Direction →	Third Column	DIRECT(1,3), DIRECT(2,3), DIRECT(3,3)

For shell and membrane elements, the first two directions are in the plane of the element and the third direction is the normal

This information is not available for beam and truss elements

Variables Passed in for Information

`T(3,3)` An array containing the direction cosines of the material orientation components relative to the element basis directions

`T(3,3)` The orientation that defines the material directions in terms of the element basis directions

`DIRECT(3,3)` The orientation that defines the material directions in terms of the global basis directions

For Continuum Elements T and DIRECT are identical

For shell and membrane elements

$$\mathrm{T}(3,3) = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

θ is the counterclockwise rotation around the normal vector that defines the orientation

Orientation is not available for beam and truss elements

Variables Passed in for Information

`TIME(1)`	Value of step time at the end of the current increment
`TIME(2)`	Value of total time at the end of the current increment
`DTIME`	Time increment
`CMNAME`	User-specified material name, left justified
`ORNAME`	User-specified local orientation name, left justified
`NUVARM`	User-specified number of user-defined output variables
`NOEL`	Element number
`NPT`	Integration point number

Variables Passed in for Information

LAYER	Layer number (for composite shells and layered solids)
KSPT	Section point number within the current layer
KSTEP	Step number
KINC	Increment number
NDI	Number of direct stress components at this point
NSHR	Number of shear stress components at this point
COORD	Coordinates at this material (integration) point

Variables Passed in for Information

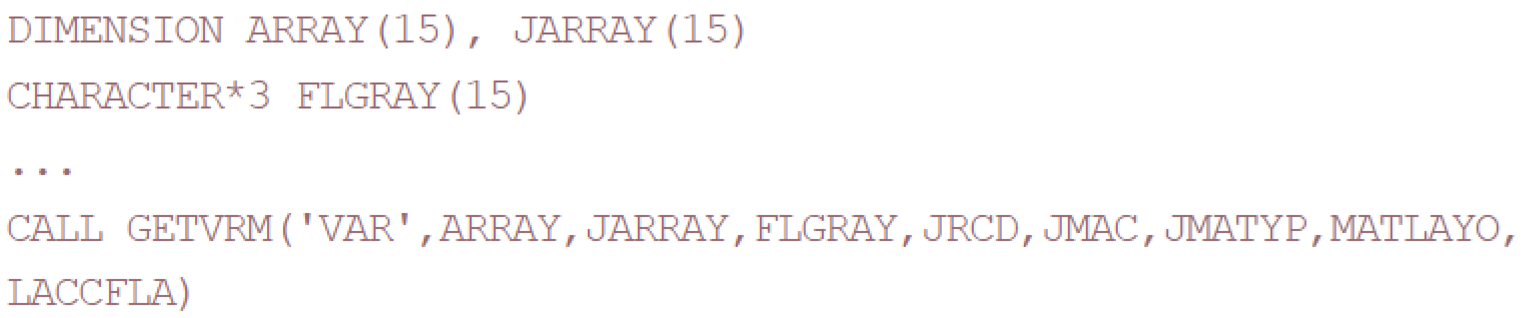
Variables that must be passed into the GETVRM utility routine

JMAC	Variable that must be passed into the GETVRM utility routine to access an output variable
JMATYP	Variable that must be passed into the GETVRM utility routine to access an output variable
MATLAYO	Variable that must be passed into the GETVRM utility routine to access an output variable
LACCFLA	Variable that must be passed into the GETVRM utility routine to access an output variable

GETVRM

Obtaining Material Point Information in an Abaqus/Standard Analysis

Utility Routine Interface

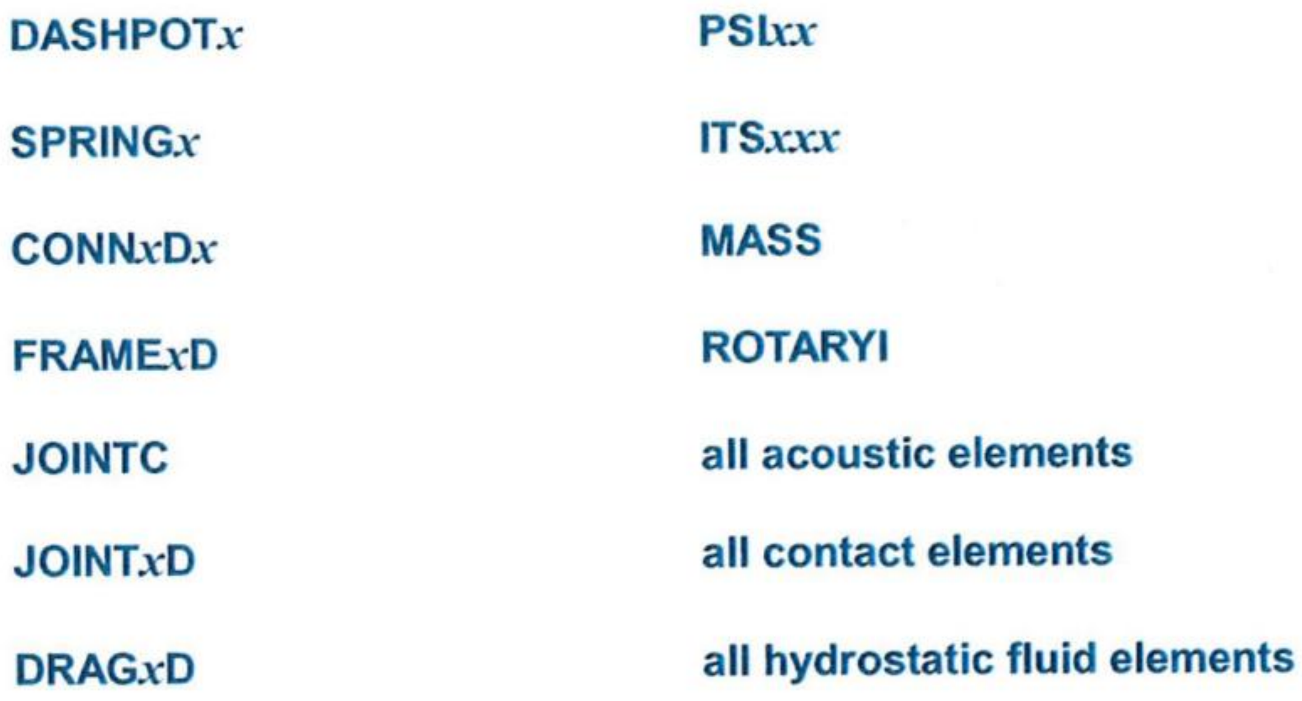


```
DIMENSION ARRAY(15), JARRAY(15)
CHARACTER*3 FLGRAY(15)
...
CALL GETVRM('VAR',ARRAY,JARRAY,FLGRAY,JRCD,JMAC,JMATYP,MATLAYO,
LACCFLA)
```

Elements supported by GETVRM

Since the GETVRM capability pertains to material point quantities, it cannot be used for most of the element types that do not require a material definition.

The following element types are, therefore, not supported:



- DASHPOT*x*
- SPRING*x*
- CONN*x*D*x*
- FRAME*x*D
- JOINTC
- JOINT*x*D
- DRAG*x*D
- PSI*xx*
- ITS*xxx*
- MASS
- ROTARYI
- all acoustic elements
- all contact elements
- all hydrostatic fluid elements

Variables to Be Provided to the Utility Routine

VAR ⟶ Output Variable Key

Variable Name	Variable Key
All stress components	**S**
ij^{th} component of stress $(i \leq j \leq 3)$	**S**ij
All principal stresses	**SP**
Minimum, intermediate, and maximum principal stresses $(SP1 \leq SP2 \leq SP3)$	**SP**n
All stress invariant components (MISES, TRESC, PRESS, INV3)	**SINV**
Signed von Mises equivalent stress	**S_MISES**
Mises equivalent stress	**MISES**

Variable Name	Variable Key
All strain components	**E**
ij^{th} component of strain $(i \leq j \leq 3)$	**E**ij
All principal strains	**EP**
Minimum, intermediate, and maximum principal strains $(EP1 \leq EP2 \leq EP3)$	**EP**n
All nominal strain components	**NE**
ij^{th} component of nominal strain $(i \leq j \leq 3)$	**NE**ij
All principal nominal strains	**NEP**
Minimum, intermediate, and maximum principal nominal strains $(NEP1 \leq NEP2 \leq NEP3)$	**NEP**n

Variables to Be Provided to the Utility Routine

The components for a requested variable

- Single index components (and requests without components) are returned in positions 1, 2, 3, etc

- Double index components (tensors) are returned in the order 11, 22, 33, 12,13, 23 for symmetric tensors, followed by 21, 31, 32 for unsymmetric tensors, such as the deformation gradient

- Three values are always returned for principal value requests, the minimum value first and maximum value third, regardless of the dimensionality of the analysis.

Variables to Be Provided to the Utility Routine

JMAC Variable that must be passed into the GETVRM utility routine to access an output variable

JMATYP Variable that must be passed into the GETVRM utility routine to access an output variable

MATLAYO Variable that must be passed into the GETVRM utility routine to access an output variable

LACCFLA Variable that must be passed into the GETVRM utility routine to access an output variable

Variables Returned from the Utility Routine

ARRAY Real array containing individual components of the output variable

JARRAY Integer array containing individual components of the output variable

FLGRAY Character array containing flags corresponding to the individual components. Flags will contain either YES, NO, or N/A (not applicable)

JRCD Return code

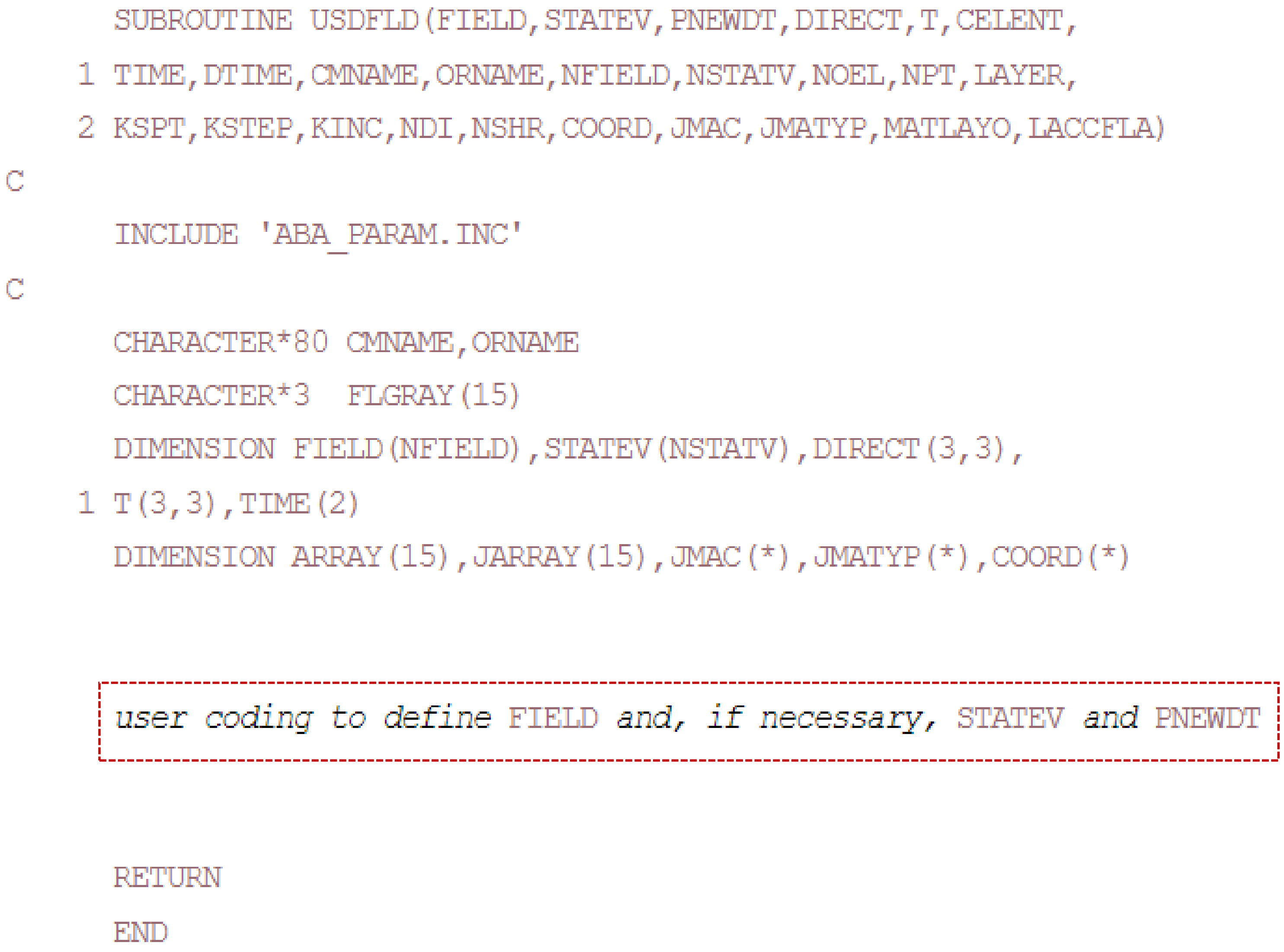
- 0 No error
- 1
  - Output request error
  - All components of the output request are zero

UVARM EXAMPLE

$$\sigma_{\mathrm{VM}} = \sqrt{\frac{1}{2}\left[\left(\sigma_{xx}-\sigma_{yy}\right)^2+\left(\sigma_{yy}-\sigma_{zz}\right)^2+\left(\sigma_{zz}-\sigma_{xx}\right)^2\right]+3\left(\tau_{xy}^2+\tau_{yz}^2+\tau_{zx}^2\right)}$$

USDFLD

Abaqus User Subroutine To Redefine Field Variables at Material Point



```
      SUBROUTINE USDFLD(FIELD,STATEV,PNEWDT,DIRECT,T,CELENT,
     1 TIME,DTIME,CMNAME,ORNAME,NFIELD,NSTATV,NOEL,NPT,LAYER,
     2 KSPT,KSTEP,KINC,NDI,NSHR,COORD,JMAC,JMATYP,MATLAYO,LACCFLA)
C
      INCLUDE 'ABA_PARAM.INC'
C
      CHARACTER*80 CMNAME,ORNAME
      CHARACTER*3  FLGRAY(15)
      DIMENSION FIELD(NFIELD),STATEV(NSTATV),DIRECT(3,3),
     1 T(3,3),TIME(2)
      DIMENSION ARRAY(15),JARRAY(15),JMAC(*),JMATYP(*),COORD(*)


      user coding to define FIELD and, if necessary, STATEV and PNEWDT


      RETURN
      END
```

USDFLD

Abaqus User Subroutines To Redefine a Field Variables at Material Point

User subroutine USDFLD is **typically** used when complex material behavior needs to be modeled, and the user does not want to develop a UMAT or VUMAT subroutine, respectively.

Allows you to define field **variables** at a material point as **functions of time or any of the available material point quantities** except the user-defined output variables UVARM and UVARMn

USDFLD or VUSDFLD is used to introduce solution-dependent material properties since such properties can easily be defined as functions of field variables

Most material properties in Abaqus can be defined as functions of field variables, f_i

USDFLD allows the user to define f_i at every integration point of an element

The subroutines have access to solution data, so $f_i(\sigma, \varepsilon, \varepsilon_{pl}, \dot{\varepsilon}, \dots);$
therefore, the material properties can be a function of the solution data.

USDFLD

Abaqus User Subroutines To Redefine a Field Variables at Material Point

Typically the user must define the dependence of material properties, such as elastic modulus or yield stress, as functions of field variables, f_i .

This can be accomplished using either tabular input or additional user subroutines

- Using tabular definition for built-in Abaqus material models
- Using other user subroutines to define the material behavior as a function of f_i .
  - CREEP
  - HETVAL
  - UEXPAN
  - UHARD
  - UHYPEL
  - UMAT
  - UMATHT
  - UTRS

E.g., field variables defined in USDFLD are passed into UMAT

The material properties defined in these subroutines are made functions of the f_i

USDFLD

Abaqus User Subroutines To Redefine a Field Variables at Material Point

The USDFLD routine is then written to define the values of f_i on an integration point-by-integration point basis.

$$f_i \begin{cases} \text{Damage to the material} \\ \text{Functionally Graded Material (FGM)} \\ \text{Bone Remodeling} \end{cases}$$

Abaqus will use linear interpolation between data points in the tabular input and will use the last available material data if f_i , is outside of the range specified—it does not extrapolate the data provided.

The range of f_i , does not have to be the same for each material property.

USDFLD

Abaqus User Subroutines To Redefine a Field Variables at Material Point

In Abaqus/Standard the USDFLD subroutine has access to material point quantities only **at the start of the increment**; thus, the solution dependence introduced in this way is explicit

The material properties are not influenced by the results obtained during the increment

Hence, the accuracy of the results depends on the size of the time increment

Therefore, the user can control the time increment in the USDFLD subroutine by means of the variable PNEWDT

USDFLD

Abaqus User Subroutines To Redefine a Field Variables at Material Point

What values for the field variables does Abaqus use?

Field variables f_i are considered nodal data by Abaqus → When Abaqus begins to calculate the element stresses and stiffness (i.e., the element loop), it interpolates the nodal values of f_i to the integration (material) points of the elements.

When subroutine USDFLD is used, however, these interpolated f_i are replaced with the values defined in the USDFLD subroutine before the material properties of an element are calculated.

Variables to Be Defined

`FIELD(NFIELD)` An array containing the field variables at the current material point.

These are passed in with the values interpolated from the nodes at the end of the current increment, as specified with initial condition definitions, **predefined field variable definitions**, or **user subroutine UFIELD**.

The updated values are used to calculate the values of **material properties** that are defined to depend on field variables and are passed into other user subroutines (CREEP, HETVAL, UEXPAN, UHARD, UHYPEL, UMAT, UMATHT, and UTRS) that are called at this material point.

The values defined by USDFLD are not stored by Abaqus

Variables That Can Be Updated

STATEV (NSTATV)

An array containing the solution-dependent state variables

These are passed in as the values at the beginning of the increment.

In all cases STATEV can be updated in this subroutine, and the updated values are passed into other user subroutines (CREEP, HETVAL, UEXPAN, UMAT, UMATHT, and UTRS) that are called at this material point

The number of state variables associated with the current material point is defined with the *DEPVAR option (keyword)

Solution-dependent state variables (SDVs) must be used in USDFLD, f_i if have any history dependence

Variables That Can Be Updated

Abaqus/Standard uses an automatic time incrementation algorithm to control the size of the time increment used in an analysis.

This algorithm allows Abaqus/Standard to reduce the time increment size when convergence is unlikely or the results are not accurate enough and to increase the time increment when convergence is easily obtained

PNEWDT Ratio of suggested new time increment to the time increment being used

If Automatic Time Incrementation Is Chosen → This variable allows you to provide input to the automatic time incrementation algorithms in Abaqus/Standard

Variables That Can Be Updated

PNEWDT is set to a large value before each call to USDFLD

IF PNEWDT is redefined to be less than 1.0

Abaqus must abandon the time increment and attempt it again with a smaller time increment

The suggested new time increment provided to the automatic time integration algorithms is PNEWDT*DTIME

where the PNEWDT used is the minimum value for all calls to user subroutines that allow redefinition of PNEWDT for this iteration.

IF PNEWDT is given a value that is greater than 1.0

(For all calls to user subroutines for this iteration and the increment converges in this iteration)

Abaqus may increase the time increment

The suggested new time increment provided to the automatic time integration algorithms is PNEWDT*DTIME

Where the PNEWDT used is the minimum value for all calls to user subroutines for this iteration.

Variables Passed in for Information

`DIRECT(3,3)` An array containing the direction cosines of the material directions in terms of the global basis directions

First Material Direction	→	First Column	DIRECT(1,1), DIRECT(2,1), DIRECT(3,1)
Second Material Direction	→	Second Column	DIRECT(1,2), DIRECT(2,2), DIRECT(3,2)
Third Material Direction	→	Third Column	DIRECT(1,3), DIRECT(2,3), DIRECT(3,3)

For shell and membrane elements, the first two directions are in the plane of the element and the third direction is the normal

This information is not available for beam and truss elements

Variables Passed in for Information

`T(3,3)` An array containing the direction cosines of the material orientation components relative to the element basis directions

`T(3,3)` The orientation that defines the material directions in terms of the element basis directions

`DIRECT(3,3)` The orientation that defines the material directions in terms of the global basis directions

For Continuum Elements T and DIRECT are identical

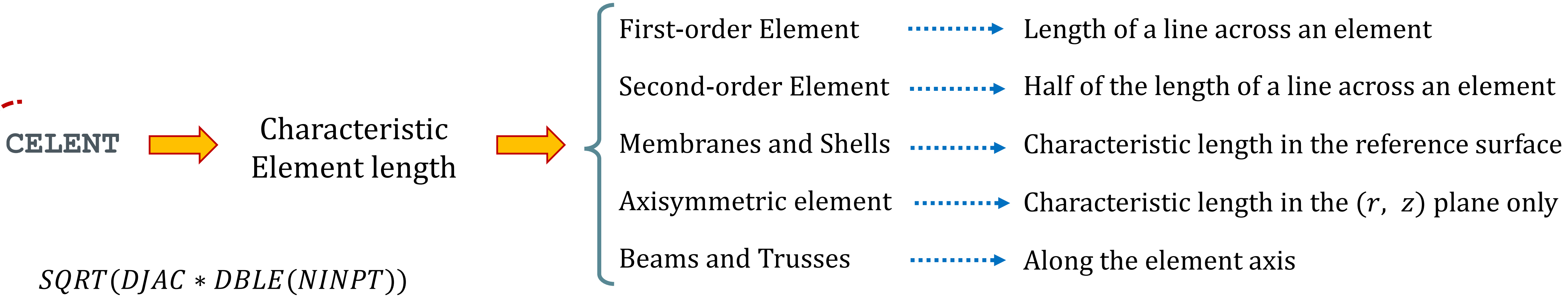
For shell and membrane elements

$$\mathrm{T}(3,3) = \begin{bmatrix} cos(\theta) & -sin(\theta) & 0 \\ sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

θ is the counterclockwise rotation around the normal vector that defines the orientation

Orientation is not available for beam and truss elements

Variables Passed in for Information



CELENT → Characteristic Element length →

- First-order Element → Length of a line across an element
- Second-order Element → Half of the length of a line across an element
- Membranes and Shells → Characteristic length in the reference surface
- Axisymmetric element → Characteristic length in the $(r,\ z)$ plane only
- Beams and Trusses → Along the element axis

$SQRT(DJAC * DBLE(NINPT))$

TIME(1) Value of **step time** at the beginning of the current increment

TIME(2) Value of **total time** at the beginning of the current increment

DTIME Time increment

Variables Passed in for Information

CMNAME	User-specified material name, left justified
ORNAME	User-specified local orientation name, left justified
NFIELD	Number of field variables defined at this material point
NSTATV	User-defined number of solution-dependent state variables
NOEL	Element number
NPT	Integration point number

Variables Passed in for Information

LAYER	Layer number (for composite shells and layered solids)
KSPT	Section point number within the current layer
KSTEP	Step number
KINC	Increment number
NDI	Number of direct stress components at this point
NSHR	Number of shear stress components at this point
COORD	Coordinates at this material point

Variables Passed in for Information

Variables that must be passed into the GETVRM utility routine

`JMAC` Variable that must be passed into the GETVRM utility routine to access an output variable

`JMATYP` Variable that must be passed into the GETVRM utility routine to access an output variable

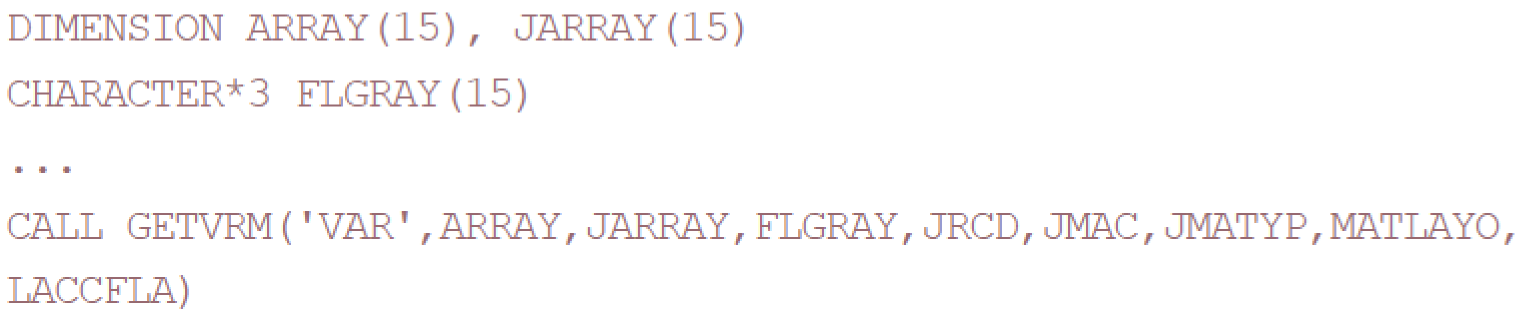
`MATLAYO` Variable that must be passed into the GETVRM utility routine to access an output variable

`LACCFLA` Variable that must be passed into the GETVRM utility routine to access an output variable

GETVRM

Obtaining Material Point Information in an Abaqus/Standard Analysis

Utility Routine Interface

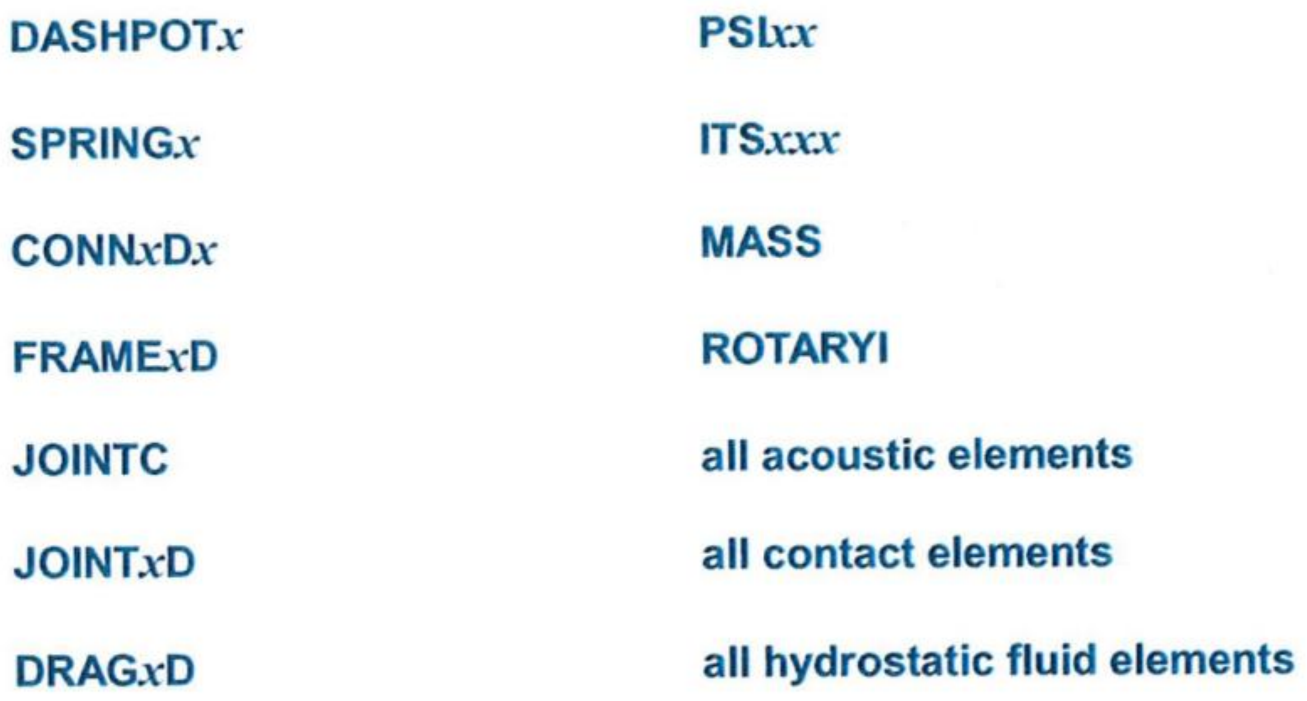


```
DIMENSION ARRAY(15), JARRAY(15)
CHARACTER*3 FLGRAY(15)
...
CALL GETVRM('VAR',ARRAY,JARRAY,FLGRAY,JRCD,JMAC,JMATYP,MATLAYO,
LACCFLA)
```

Elements supported by GETVRM

Since the GETVRM capability pertains to material point quantities, it cannot be used for most of the element types that do not require a material definition.

The following element types are, therefore, not supported:



DASHPOT*x*

SPRING*x*

CONN*x*D*x*

FRAME*x*D

JOINTC

JOINT*x*D

DRAG*x*D

PSI*xx*

ITS*xxx*

MASS

ROTARYI

all acoustic elements

all contact elements

all hydrostatic fluid elements

Variables to Be Provided to the Utility Routine

`VAR` → Output Variable Key

Variable Name	Variable Key
All stress components	**S**
ij^{th} component of stress $(i \le j \le 3)$	**S**ij
All principal stresses	**SP**
Minimum, intermediate, and maximum principal stresses $(SP1 \le SP2 \le SP3)$	**SP**n
All stress invariant components (MISES, TRESC, PRESS, INV3)	**SINV**
Signed von Mises equivalent stress	**S_MISES**
Mises equivalent stress	**MISES**

Variable Name	Variable Key
All strain components	**E**
ij^{th} component of strain $(i \le j \le 3)$	**E**ij
All principal strains	**EP**
Minimum, intermediate, and maximum principal strains $(EP1 \le EP2 \le EP3)$	**EP**n
All nominal strain components	**NE**
ij^{th} component of nominal strain $(i \le j \le 3)$	**NE**ij
All principal nominal strains	**NEP**
Minimum, intermediate, and maximum principal nominal strains $(NEP1 \le NEP2 \le NEP3)$	**NEP**n

Variables to Be Provided to the Utility Routine

The components for a requested variable

Single index components (and requests without components) are returned in positions 1, 2, 3, etc

Double index components (tensors) are returned in the order 11, 22, 33, 12,13, 23 for symmetric tensors, followed by 21, 31, 32 for unsymmetric tensors, such as the deformation gradient

Three values are always returned for principal value requests, the minimum value first and maximum value third, regardless of the dimensionality of the analysis.

Variables to Be Provided to the Utility Routine

JMAC Variable that must be passed into the GETVRM utility routine to access an output variable

JMATYP Variable that must be passed into the GETVRM utility routine to access an output variable

MATLAYO Variable that must be passed into the GETVRM utility routine to access an output variable

LACCFLA Variable that must be passed into the GETVRM utility routine to access an output variable

Variables Returned from the Utility Routine

ARRAY Real array containing individual components of the output variable

JARRAY Integer array containing individual components of the output variable

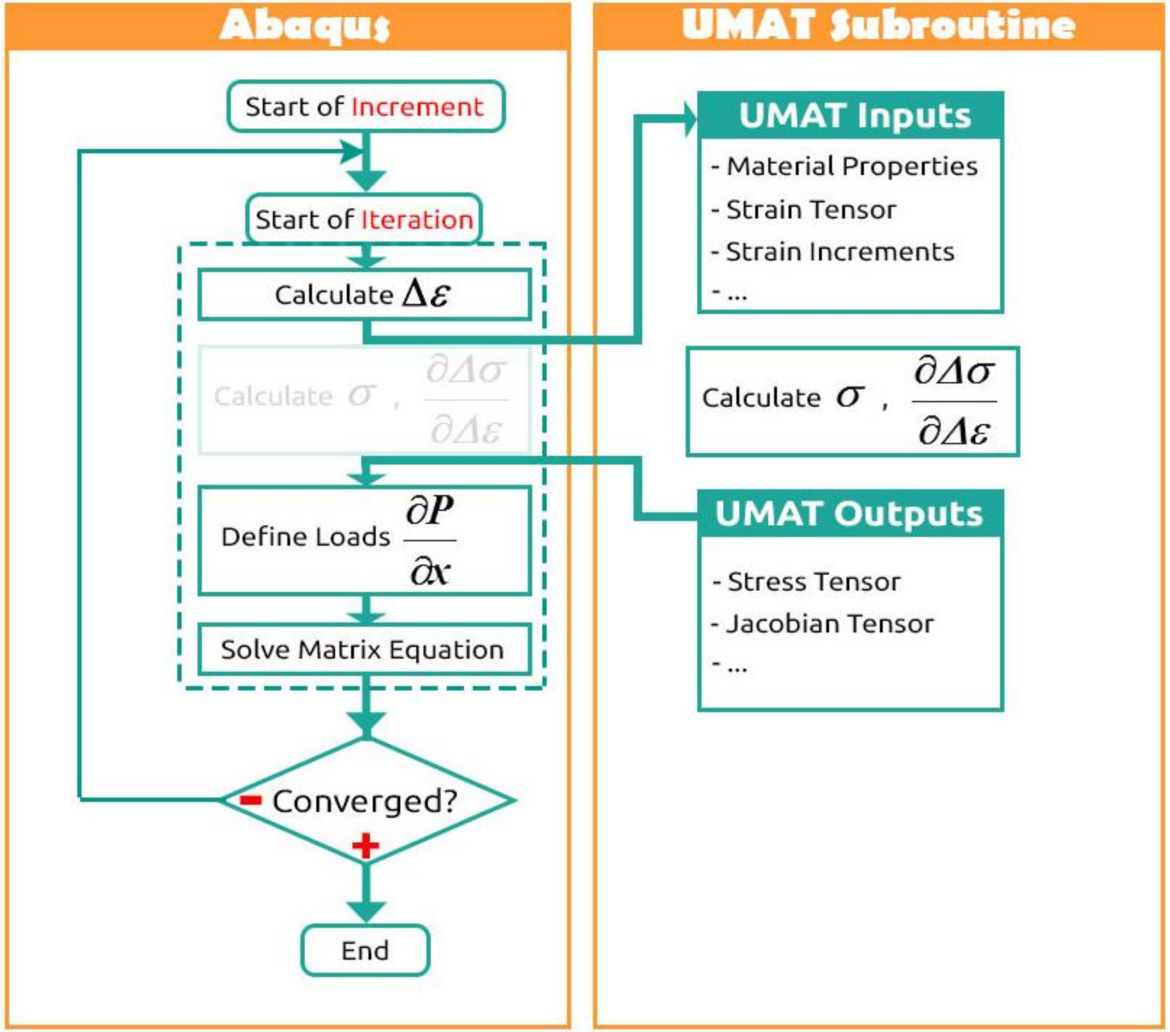
FLGRAY Character array containing flags corresponding to the individual components.
Flags will contain either YES, NO, or N/A (not applicable)

JRCD Return code

- 0 No error
- 1
  - Output request error
  - All components of the output request are zero

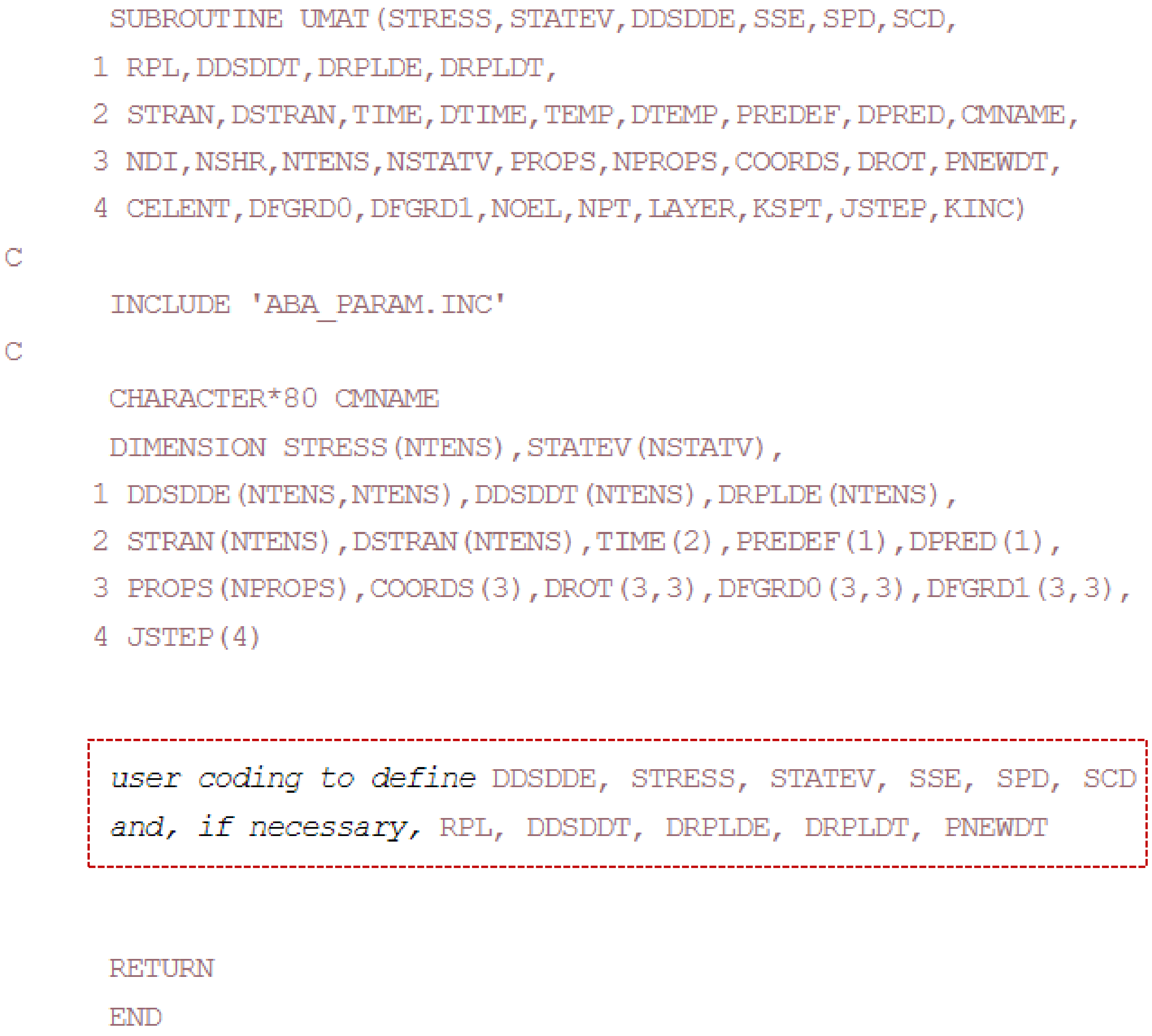
UMAT

Abaqus User Subroutines To Define a Material's Mechanical Behavior


Abaqus
Start of Increment
Start of Iteration
Calculate $\Delta\varepsilon$
Calculate σ , $\frac{\partial\Delta\sigma}{\partial\Delta\varepsilon}$
Define Loads $\frac{\partial P}{\partial x}$
Solve Matrix Equation
- Converged?
+
End
UMAT Subroutine
UMAT Inputs
- Material Properties
- Strain Tensor
- Strain Increments
- ...
Calculate σ , $\frac{\partial\Delta\sigma}{\partial\Delta\varepsilon}$
UMAT Outputs
- Stress Tensor
- Jacobian Tensor
- ...

UMAT

User Subroutine Interface



```
      SUBROUTINE UMAT(STRESS,STATEV,DDSDDE,SSE,SPD,SCD,
     1 RPL,DDSDDT,DRPLDE,DRPLDT,
     2 STRAN,DSTRAN,TIME,DTIME,TEMP,DTEMP,PREDEF,DPRED,CMNAME,
     3 NDI,NSHR,NTENS,NSTATV,PROPS,NPROPS,COORDS,DROT,PNEWDT,
     4 CELENT,DFGRD0,DFGRD1,NOEL,NPT,LAYER,KSPT,JSTEP,KINC)
C
      INCLUDE 'ABA_PARAM.INC'
C
      CHARACTER*80 CMNAME
      DIMENSION STRESS(NTENS),STATEV(NSTATV),
     1 DDSDDE(NTENS,NTENS),DDSDDT(NTENS),DRPLDE(NTENS),
     2 STRAN(NTENS),DSTRAN(NTENS),TIME(2),PREDEF(1),DPRED(1),
     3 PROPS(NPROPS),COORDS(3),DROT(3,3),DFGRD0(3,3),DFGRD1(3,3),
     4 JSTEP(4)


      user coding to define DDSDDE, STRESS, STATEV, SSE, SPD, SCD
      and, if necessary, RPL, DDSDDT, DRPLDE, DRPLDT, PNEWDT


      RETURN
      END
```

UMAT

Variables passed in for information

STRAN(NTENS)	An array containing the total (mechanical) strains at the beginning of the increment	Engineering Shear Components
DSTRAN(NTENS)	Array of (mechanical) strain increments	
TIME(1)	Value of step time at the beginning of the current increment or frequency	
TIME(2)	Value of total time at the beginning of the current increment	
DTIME	Time increment	
TEMP	Temperature at the start of the increment	
DTEMP	Increment of temperature	
PREDEF	Array of interpolated values of predefined field variables	
DPRED	Array of increments of predefined field variables	
CMNAME	User-defined material name	To avoid conflict, you should not use “ABQ_” as the leading string for CMNAME

UMAT

Variables passed in for information

NTENS=NDI+NSHR — Size of the stress or strain component array

- NDI — Number of direct stress components at this point
- NSHR — Number of engineering shear stress components at this point

Plane Stress: 3 Axisymmetric, and (Generalized) Plane Strain: 4 3D Stress: 6

NSTATV	Number of solution-dependent state variables	
PROPS(NPROPS)	Array of material constants	
NPROPS	Number of material constants	
COORDS	An array containing the coordinates of this point	
DROT(3,3)	Rotation increment matrix	stress and strain components are already rotated by this amount before UMAT is called
CELENT	Characteristic element length	First-order: length of a line across an element Second-order: Half of the First-order
DFGRD0(3,3)	Array containing the deformation gradient at the beginning of the increment	
DFGRD1(3,3)	Array containing the deformation gradient at the end of the increment	Identity matrix if nonlinear geometric effects are not included in the step definition

UMAT

Variables passed in for information

NOEL	Element number
NPT	Integration point number
LAYER	Layer number (for composite shells and layered solids)
KSPT	Section point number within the current layer
JSTEP(1)	Step number
JSTEP(2)	Procedure type key
JSTEP(3)	1 if NLGEOM=YES for the current step; 0 otherwise
JSTEP(4)	1 if current step is a linear perturbation procedure; 0 otherwise
KINC	Increment number

UMAT

Variables to be defined

DDSDDE(NTENS,NTENS) Jacobian matrix of the constitutive model

f: vector-valued function of several variables

$$\mathbf{J} = \begin{bmatrix} \dfrac{\partial \mathbf{f}}{\partial x_1} & \cdots & \dfrac{\partial \mathbf{f}}{\partial x_n} \end{bmatrix} = \frac{\partial(f_1, \ldots, f_m)}{\partial(x_1, \ldots, x_n)} = \begin{bmatrix} \dfrac{\partial f_1}{\partial x_1} & \cdots & \dfrac{\partial f_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \dfrac{\partial f_m}{\partial x_1} & \cdots & \dfrac{\partial f_m}{\partial x_n} \end{bmatrix}$$

Consistent Jacobian base on Constitutive Laws

Total-form Constitutive Laws $\xrightarrow{\text{Exact Consistent Jacobian}}$

$$\delta(J\boldsymbol{\sigma}) = J\,(\mathbf{C} : \delta\mathbf{D} + \delta\mathbf{W}.\boldsymbol{\sigma} - \boldsymbol{\sigma}.\delta\mathbf{W})$$

$$\delta\mathbf{D} = sym(\delta\mathbf{F}\,.\,\mathbf{F}^{-1})$$

$$\delta\mathbf{W} = asym(\delta\mathbf{F}\,.\,\mathbf{F}^{-1})$$

total-form constitutive laws relate current stress states directly to current strain states or deformation measures

more stable numerically and less prone to accumulation of errors over time

Rate-form constitutive laws express relationships between stress rates and strain rates, offering advantages in handling path-dependent material behavior and large deformations

Rate-form Constitutive Laws $\xrightarrow{\text{Exact Consistent Jacobian}}$

$\Delta\,(J\boldsymbol{\sigma})$ are the Kirchhoff stress increments,

$$\mathbf{C} = \frac{1}{J}\,\frac{\partial\Delta(J\boldsymbol{\sigma})}{\partial\Delta\boldsymbol{\varepsilon}}$$

Determinant of the Deformation Gradient

$\Delta\boldsymbol{\varepsilon}$ are the strain increments.

Rate-form constitutive laws establish relationships between rates of stress and rates of strain or deformation, providing a differential framework that describes how stress evolves with changing deformation states.

Rate-form Constitutive Laws

The mathematical framework of rate-form constitutive laws requires careful consideration of objectivity, particularly when dealing with finite deformations and rotations. Since stress rates must be frame-indifferent to ensure physical consistency, various objective stress rates have been developed to maintain this requirement

The choice of objective stress rate significantly impacts the material model's behavior and numerical performance. Common objective stress rates include the Truesdell rate, the Green-Naghdi rate, and the Zaremba-Jaumann rate of the Cauchy stress, each with distinct mathematical properties and applications

Rate-form constitutive laws excel in capturing certain types of material behavior that are difficult to represent with total-form approaches. They naturally accommodate path-dependent phenomena, rate-sensitive materials, and complex loading histories where the material response depends on the sequence and rate of deformation rather than just the final state

UMAT

Variables to be defined

An incorrect definition of the material Jacobian affects only the convergence rate; the results (if obtained) are unaffected

DDSDDE(NTENS,NTENS)

Determinant of the Deformation Gradient

$$\mathbf{C} = \frac{1}{J}\frac{\partial \Delta\left(J\boldsymbol{\sigma}\right)}{\partial \Delta\boldsymbol{\varepsilon}}$$

$\Delta\left(J\boldsymbol{\sigma}\right)$ are the Kirchhoff stress increments,

$\Delta\boldsymbol{\varepsilon}$ are the strain increments.

Loss of quadratic convergence may occur

If the **volume change is small**, the Jacobian matrix can be approximated as

- For small-deformation problems (e.g., linear elasticity)
- large-deformation problems with small volume changes (e.g., metal plasticity)

$$\mathbf{C} = \frac{\partial \Delta\boldsymbol{\sigma}}{\partial \Delta\boldsymbol{\varepsilon}}$$

$\Delta\boldsymbol{\sigma}$: Cauchy stress increments

For viscoelastic behavior in the frequency domain, the Jacobian matrix must be dimensioned as DDSDDE(NTENS,NTENS,2)

DDSDDE(NTENS,NTENS,1)	Stiffness contribution (storage modulus)
DDSDDE(NTENS,NTENS,2)	Damping contribution (loss modulus)

UMAT

Variables to be defined

$$\overset{\text{Kirchhoff stress}}{\tau} = \underset{\text{Determinant of the Deformation Gradient}}{J}\,\sigma$$

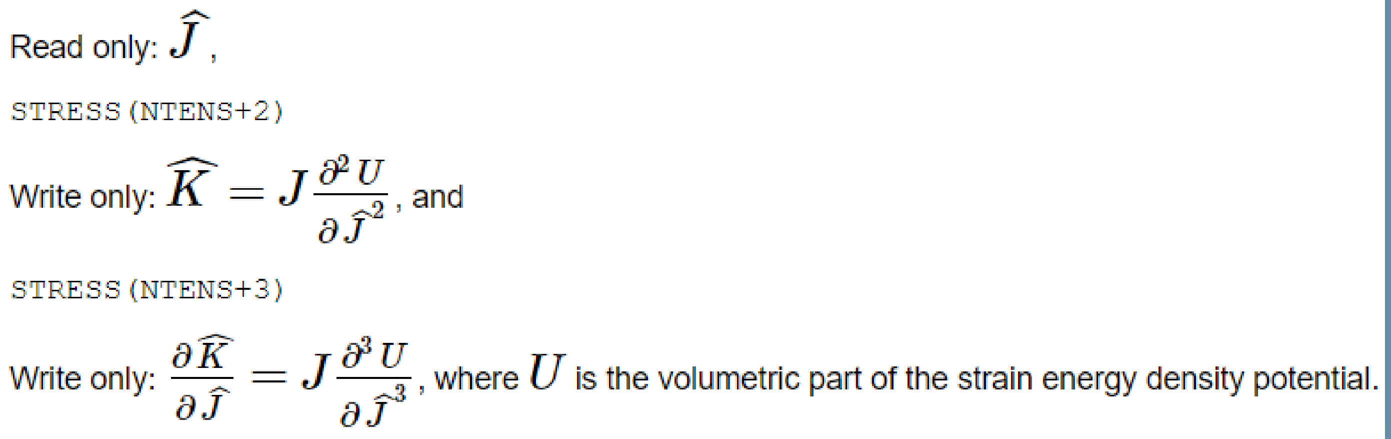
STRESS(NTENS)

This array is passed in as the “true” (Cauchy) stress tensor at the beginning of the increment and must be updated in this routine to be the stress tensor at the end of the increment

In finite-strain problems the stress tensor **has already been rotated** to account for rigid body motion in the increment before UMAT is called, so that only the **corotational** part of the stress integration should be done in UMAT.

Hybrid formulation

- Incremental (default)
- Total
- Incompressible



Read only: $\hat{J}$,

```
STRESS(NTENS+2)
```

Write only: $\widehat{K} = J\frac{\partial^2 U}{\partial \hat{J}^2}$, and

```
STRESS(NTENS+3)
```

Write only: $\frac{\partial \widehat{K}}{\partial \hat{J}} = J\frac{\partial^3 U}{\partial \hat{J}^3}$, where U is the volumetric part of the strain energy density potential.

UMAT

Variables to be defined

STATEV(NSTATV)

Solution-dependent State Variables →

DepVar: In Property
SDV: In Field Output
STATEV: In UMAT

They are values that can be defined to evolve with the solution of an analysis

These are passed in as the values at the beginning of the increment unless they are updated in user subroutines USDFLD or UEXPAN, in which case the updated values are passed in. In all cases STATEV must be returned as the values at the end of the increment

SSE Specific Elastic Strain Energy

SPD Specific Plastic Dissipation

SCD Specific Creep Dissipation

They are used for energy output

UMAT

Variables to be defined

Only in a fully **coupled thermal-stress** or a coupled **thermal-electrical-structural** analysis

RPL	Volumetric heat generation per unit time at the end of the increment caused by mechanical working of the material
DDSDDT(NTENS)	Variation of the stress increments with respect to the temperature
DRPLDE(NTENS)	Variation of RPL with respect to the strain increments
DRPLDT	Variation of RPL with respect to the temperature

UMAT

Variables That Can Be Updated

PNEWDT Ratio of suggested new time increment to the time increment being used

This variable allows you to provide input to the automatic time incrementation algorithms in Abaqus/Standard

The suggested new time increment provided to the automatic time integration algorithms is PNEWDT × DTIME, where the PNEWDT used is the minimum value for all calls to user subroutines that allow redefinition of PNEWDT for this iteration.

Formulation Approach

Total Lagrangian Approach

For the total Lagrangian approach, the discrete equations are formulated with respect to the reference configuration. The **independent variables** are t and $\mathbf{X} = \chi(\mathbf{x})$ and the **dependent variable** is displacement $u(X, t)$.

Updated Lagrangian Approach

For the updated Lagrangian approach, the discrete equations are formulated in the **current configuration**, which is assumed to be the new reference configuration. The stress is measured by the **Cauchy stress**.
The **dependent variables** are chosen to be the stress $\sigma(\mathbf{X}, t)$ and the velocity $v(\mathbf{X}, t)$.
In developing the updated Lagrangian formulation, we will sometimes need the dependent variables to be expressed in terms of the Eulerian coordinates.

Eulerian Approach

In an Eulerian formulation, the nodes are fixed in space and the **dependent variables** are functions of the Eulerian spatial coordinate x and the time t. The stress measure is the Cauchy stress $\sigma(\mathbf{x}, t)$, the measure of deformation is the **rate-of-deformation** $\nabla v(\mathbf{x}, t)$, and the motion will be described by the velocity $v(\mathbf{x}, t)$.

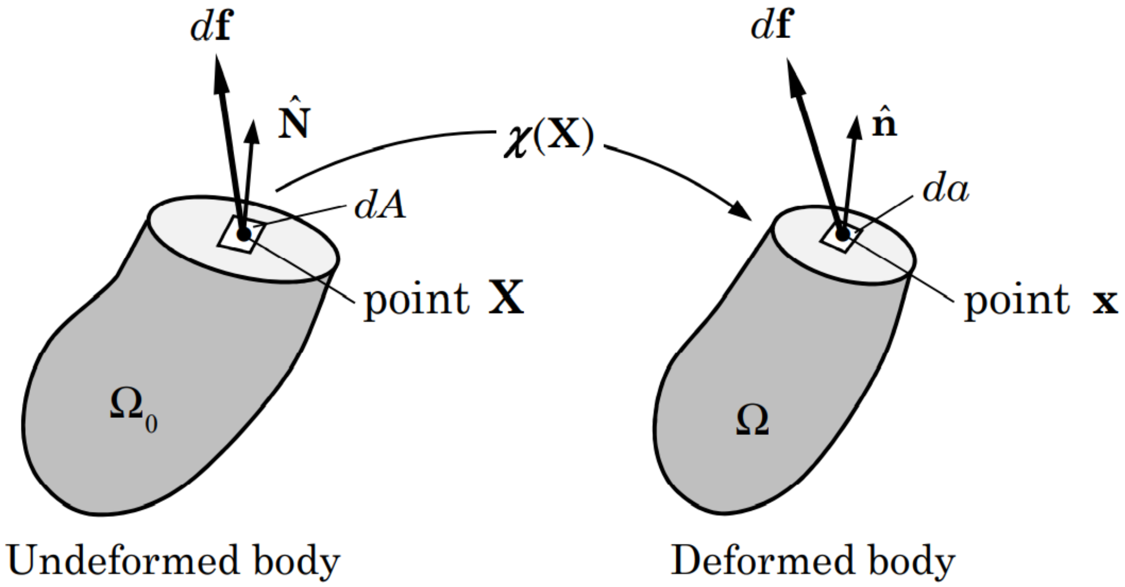
Description of Motion

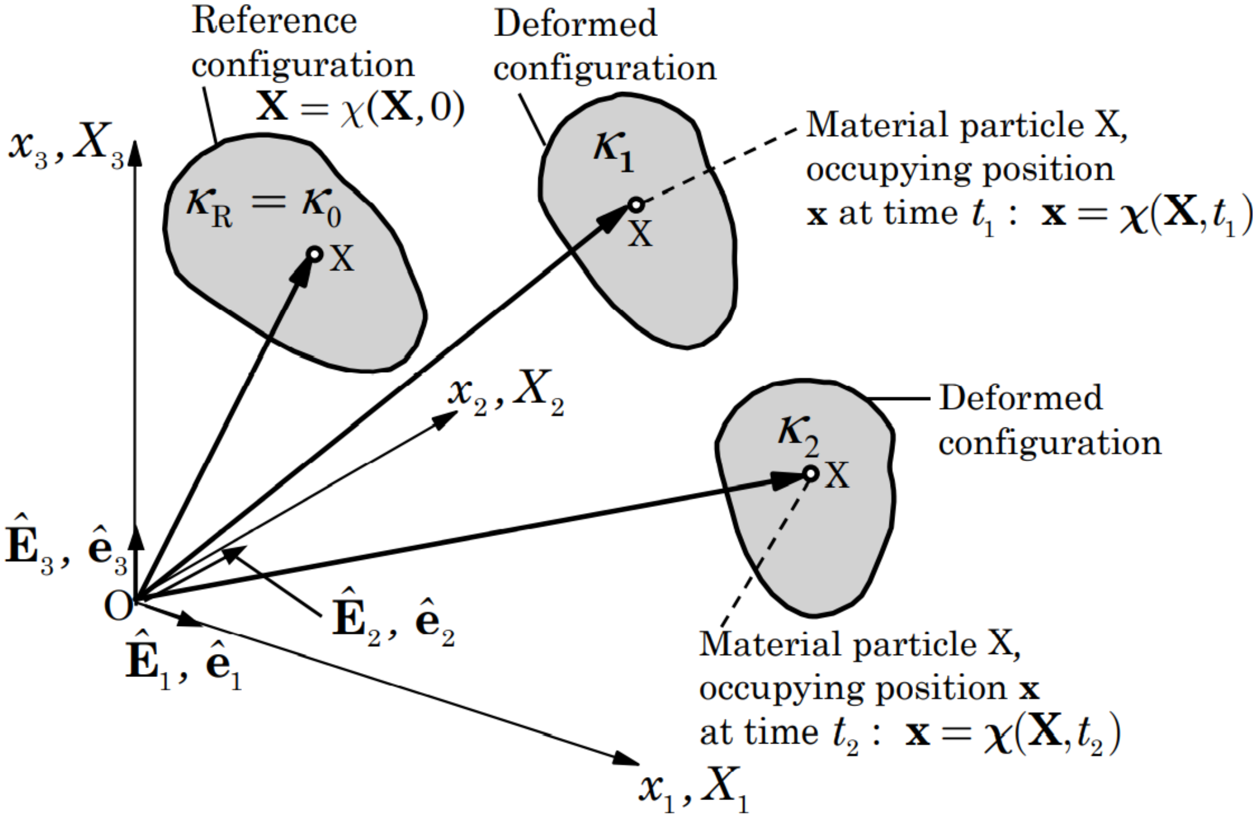
$$\mathbf{x} = \chi(\mathbf{X}, t), \qquad \chi(X, 0) = X$$

$$[F] = \begin{bmatrix} \frac{\partial x_1}{\partial X_1} & \frac{\partial x_1}{\partial X_2} & \frac{\partial x_1}{\partial X_3} \\ \frac{\partial x_2}{\partial X_1} & \frac{\partial x_2}{\partial X_2} & \frac{\partial x_2}{\partial X_3} \\ \frac{\partial x_3}{\partial X_1} & \frac{\partial x_3}{\partial X_2} & \frac{\partial x_3}{\partial X_3} \end{bmatrix}$$

$$d\mathbf{x} = \mathbf{F} \,.\, d\mathbf{X} \qquad \mathbf{F} = (\nabla_0 \mathbf{x}) = \frac{\partial \mathbf{x}\,(\mathbf{X}, t)}{\partial \mathbf{X}}$$

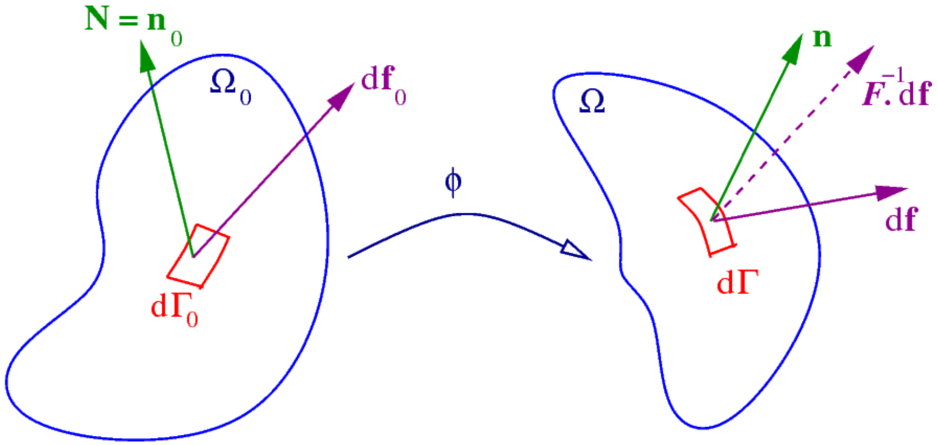
$$J = det(\mathbf{F}\,)$$





Reference configuration
$\mathbf{X} = \chi(\mathbf{X}, 0)$
Deformed configuration
x_3, X_3
κ_1
Material particle X, occupying position $\mathbf{x}$ at time t_1 : $\mathbf{x} = \chi(\mathbf{X}, t_1)$
$\kappa_R = \kappa_0$
X
X
x_2, X_2
Deformed configuration
κ_2
X
$\hat{\mathbf{E}}_3, \hat{\mathbf{e}}_3$
$\hat{\mathbf{E}}_2, \hat{\mathbf{e}}_2$
O
$\hat{\mathbf{E}}_1, \hat{\mathbf{e}}_1$
Material particle X, occupying position $\mathbf{x}$ at time t_2 : $\mathbf{x} = \chi(\mathbf{X}, t_2)$
x_1, X_1

Measure of Stress



$\boldsymbol{\sigma}$: Cauchy Stress (True Stress) is defined to be the **current** force per unit **deformed area.**

$$\boldsymbol{\sigma} = \boldsymbol{\sigma}^T$$

P: First Piola-Kirchhoff stress tensor (known as the Lagrangian stress tensor or transpose of Nominal stress) is defined to be the **current** force per unit **undeformed area.**

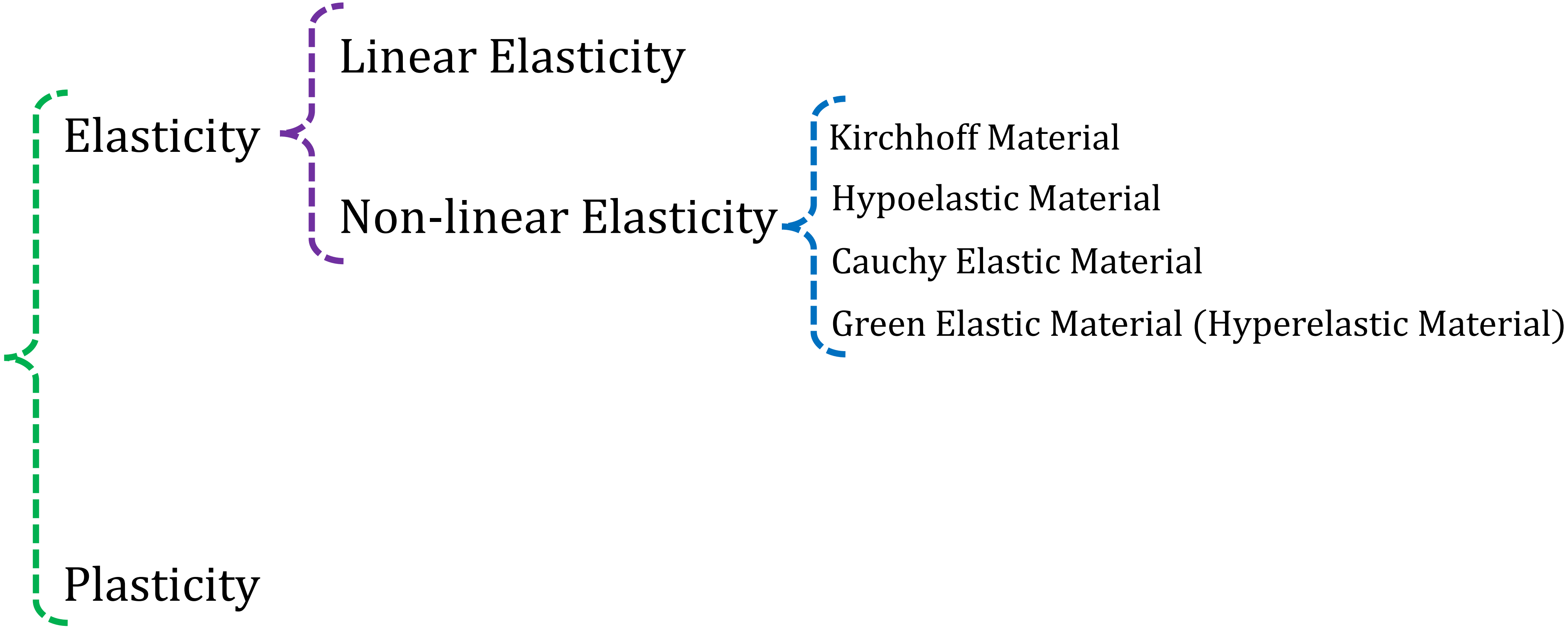
S: Second Piola-Kirchhoff stress is defined to be the **initial** (transformed current) force per unit **undeformed area.**

$$\mathbf{S} = \mathbf{S}^{\mathrm{T}}$$

$\boldsymbol{\tau}$: Kirchhoff stress $\begin{cases} \tau = J\ \sigma & \text{Cauchy Stress Scaled by The Determinant of The Jacobian} \\ \tau = \mathbf{F}.\mathbf{S}.\mathbf{F}^{\mathrm{T}} & \text{Push Forward of The Second Piola-Kirchhoff Stress} \end{cases}$

Constitutive Models

Stress As A Function Of The Deformation History Of The Body



- Elasticity
  - Linear Elasticity
  - Non-linear Elasticity
    - Kirchhoff Material
    - Hypoelastic Material
    - Cauchy Elastic Material
    - Green Elastic Material (Hyperelastic Material)
- Plasticity

All tensor quantities are defined in the corotational coordinate system that rotates with the material point

Non-linear Elasticity

Kirchhoff Material

Second Piola-Kirchhoff stress

Lagrangian (Green) strain

$$S_{ij} = C_{ijkl} E_{kl} \qquad \mathbf{S} = \mathbf{C} : \mathbf{E}$$

Elastic Moduli (Stiffness Tensor)

Hypoelastic Material

Rate of Cauchy stress is related to the rate-of-deformation

objective rate of the Cauchy stress

rate-of-deformation

$$\boldsymbol{\sigma}^{\nabla} = \mathbf{f}(\boldsymbol{\sigma}, \mathbf{D}) \Rightarrow \boldsymbol{\sigma}^{\nabla} = \mathbf{C} : \mathbf{D}$$

objective function

incrementally linear and reversible

Depend on stress

Cauchy Elastic Material

no dependence on the history of the motion

$$\boldsymbol{\sigma} = \boldsymbol{G}(\mathbf{F})$$

Green Elastic Material (Hyperelastic Material)

is path-independent and fully reversible where the stress is derived from a strain (or stored) energy potential

$$\mathbf{S} = 2 \frac{\partial \psi(\mathbf{C})}{\partial \mathbf{C}}$$

Right Cauchy-Green Deformation Tensor

Objective Stress Rates

The rate of change of the internal virtual work is required for use in the Newton (Newton–Raphson) Method

Solver	Element Type	Constitutive Model	Objective Rate
Abaqus/Standard	Solid (Continuum)	All built-in and user-defined materials	Jaumann
	Structural (Shells, Membranes, Beams, Trusses)	All built-in and user-defined materials	Green- Naghdi
Abaqus/Explicit	Solid (Continuum)	All except hyperelastic, viscoelastic, brittle cracking, and VUMAT	Jaumann
	Solid (Continuum)	Hyperelastic, viscoelastic, brittle cracking, and VUMAT	Green- Naghdi
	Structural (Shells, Membranes, Beams, Trusses)	All built-in and user-defined materials	Green- Naghdi

$$\frac{d^{\nabla J}}{dt}(J\boldsymbol{\sigma}) = \frac{d}{dt}(J\boldsymbol{\sigma}) - J(\mathbf{W}\cdot\boldsymbol{\sigma} - \boldsymbol{\sigma}\cdot\mathbf{W})$$

$$\boldsymbol{\tau}^{\nabla J} = \mathbf{C}^{\tau J} : \mathbf{D}$$

Change of stress due to rotation

$$\frac{d}{dt}(J\boldsymbol{\sigma}) = \mathbf{C}'\underset{\downarrow}{:}\mathbf{D} + J(\mathbf{W}.\overset{\uparrow}{\boldsymbol{\sigma}} - \boldsymbol{\sigma}.\mathbf{W})$$

Rate of change due to material response

Corotational Derivatives

Most General Form Of Linearized Material Behavior

Stiffness Tensor

$$\mathbf{S} = \mathbf{C} : \mathbf{E} \quad \Rightarrow \quad J\mathbf{F}^{-1}.\boldsymbol{\sigma}.\mathbf{F}^{-T} = \mathbf{C} : \mathbf{E} \quad \Rightarrow \quad J\boldsymbol{\sigma} = \mathbf{F}\,.\,(\mathbf{C} : \mathbf{E})\,.\,\mathbf{F}^{T}$$

$$\frac{d}{dt}(J\boldsymbol{\sigma}) = \dot{\mathbf{F}}\,.\,(\mathbf{C} : \mathbf{E})\,.\,\mathbf{F}^{T} + \mathbf{F}\,.\left(\mathbf{C} : \dot{\mathbf{E}}\right).\,\mathbf{F}^{T} + \mathbf{F}\,.\,(\mathbf{C} : \mathbf{E})\,.\,\dot{\mathbf{F}}^{T} \qquad + \qquad \boxed{\nabla\mathbf{v} = \mathbf{L} = \dot{\mathbf{F}}\mathbf{F}^{-1}}$$

$$\frac{d}{dt}(J\boldsymbol{\sigma}) = \mathbf{L}\,.\,(J\boldsymbol{\sigma}) + (J\boldsymbol{\sigma}).\,\mathbf{L}^{T} + \mathbf{F}\,.\,(\mathbf{C} : (\mathbf{F}^{T}.\,\mathbf{D}\,.\,\mathbf{F})\mathbf{E}).\,\mathbf{F}^{T}$$

Lie Derivative

$$\underbrace{\frac{d}{dt}(J\boldsymbol{\sigma}) - \mathbf{L}\,.\,(J\boldsymbol{\sigma}) - (J\boldsymbol{\sigma}).\,\mathbf{L}^{T}}_{\frac{d^{\nabla}}{dt}(J\boldsymbol{\sigma})} = \underbrace{(\mathbf{F}\,.\,\mathbf{F}.\,\mathbf{C}\,.\,\mathbf{F}^{T}.\,\mathbf{F}^{T})}\, : \mathbf{D} \quad \Rightarrow \quad \frac{d^{\nabla}}{dt}(J\boldsymbol{\sigma}) = \frac{d}{dt}(J\boldsymbol{\sigma}) - \mathbf{L}\,.\,(J\boldsymbol{\sigma}) - (J\boldsymbol{\sigma}).\,\mathbf{L}^{T} = \mathbf{C}' : \mathbf{D}$$

$\mathbf{C}'$: Rigid Body Rotation Of The Stiffness Tensor

Jaumann Derivative

$$\boxed{\frac{d^{\nabla J}}{dt}(J\boldsymbol{\sigma}) = \frac{d}{dt}(J\boldsymbol{\sigma}) - J(\mathbf{W}\,.\,\boldsymbol{\sigma} - \boldsymbol{\sigma}\,.\,\mathbf{W}) = \mathbf{C}' : \mathbf{D}}$$

Corotational Derivatives

$$\begin{cases} \mathbf{D} = \dfrac{1}{2}\left(\dfrac{\partial \mathbf{v}}{\partial x} + \left[\dfrac{\partial \mathbf{v}}{\partial x}\right]^T\right) \\ \mathbf{W} = \dfrac{1}{2}\left(\dfrac{\partial \mathbf{v}}{\partial x} - \left[\dfrac{\partial \mathbf{v}}{\partial x}\right]^T\right) \end{cases} \Rightarrow \quad \dot{\mathbf{e}}_\alpha = \mathbf{W}\,.\,\mathbf{e}_\alpha \;\big/\; \dot{\mathbf{e}}_\alpha = \mathbf{\Omega}\,.\,\mathbf{e}_\alpha$$

$$\mathbf{F} = \mathbf{U}\,.\,\mathbf{R} \longrightarrow \mathbf{\Omega} = \dot{\mathbf{R}}\,.\,\mathbf{R}^T$$

Rigid Body Rotation In The Polar Decomposition Of The Deformation Gradient

$$\mathbf{T} = T^{\alpha\beta}\mathbf{e}_\alpha\,\mathbf{e}_\beta^T \Rightarrow \dot{\mathbf{T}} = \dot{T}^{\alpha\beta}\mathbf{e}_\alpha\,\mathbf{e}_\beta^T + \underbrace{T^{\alpha\beta}\dot{\mathbf{e}}_\alpha\,\mathbf{e}_\beta^T + T^{\alpha\beta}\mathbf{e}_\alpha\,\dot{\mathbf{e}}_\beta^T}_{} . \Rightarrow \begin{cases} \mathbf{T}^{\nabla J} = \dot{\mathbf{T}} - \mathbf{W}\cdot\mathbf{T} + \mathbf{T}\cdot\mathbf{W} & \text{Jaumann} \\ \mathbf{T}^{\nabla G} = \dot{\mathbf{T}} - \mathbf{\Omega}\cdot\mathbf{T} + \mathbf{T}\cdot\mathbf{\Omega} & \text{Green-Naghdi} \end{cases}$$

Rate Associated With The Constitutive Response

Caused By The Rigid Body Spin

Corotational Rate

$$\frac{d^{\nabla J}}{dt}(J\boldsymbol{\sigma}) = \frac{d}{dt}(J\boldsymbol{\sigma}) - J\left(\mathbf{W}\cdot\boldsymbol{\sigma} - \boldsymbol{\sigma}\cdot\mathbf{W}\right).$$

The Principle of Virtual Displacement

“virtual” work rate

$$\iiint_{\Omega} \boldsymbol{\sigma} : \nabla(\delta \mathbf{v})\, dv = \iiint_{\Omega} \mathbf{f} \,.\, \delta\mathbf{v}\, dv + \oiint_{\Gamma} \mathbf{t} \,.\, \delta\mathbf{v}\, ds$$

Rate-of-deformation

$$(\nabla \mathbf{v}) = \mathbf{d} + \mathbf{w} \begin{cases} \mathbf{d} = \dfrac{1}{2}\left[(\nabla \mathrm{v})^{T} + (\nabla \mathrm{v})\right] \\ \mathbf{w} = \dfrac{1}{2}\left[(\nabla \mathrm{v})^{T} - (\nabla \mathrm{v})\right] \end{cases}$$

Rate-of-spin

$$\iiint_{\Omega} \boldsymbol{\sigma} : \delta\mathbf{d}\, dv = \iiint_{\Omega} \mathbf{f} \,.\, \delta\mathbf{v}\, dv + \oiint_{\Gamma} \mathbf{t} \,.\, \delta\mathbf{v}\, ds$$

$$\boxed{\nabla \mathbf{v} = \mathbf{L} = \dot{\mathbf{F}}\mathbf{F}^{-1}}$$

$$\iiint_{\Omega} \boldsymbol{\sigma} : \delta\left(\frac{1}{2}\left[\left(\dot{\mathbf{F}}\mathbf{F}^{-1}\right) + \left(\dot{\mathbf{F}}\mathbf{F}^{-1}\right)^{T}\right]\right) dv = \iiint_{\Omega} \mathbf{f} \,.\, \delta\mathbf{v}\, dv + \oiint_{\Gamma} \mathbf{t} \,.\, \delta\mathbf{v}\, ds$$

The Principle of Virtual Displacement

For initial volume and area

$$\int_{\Omega_0} J\boldsymbol{\sigma} : \boldsymbol{\nabla}(\delta\mathbf{V})\,dV - \left(\int_{\Omega_0} \mathbf{f}^0\cdot\delta\mathbf{V}\,dV + \oint_{\Gamma_0}\mathbf{t}^0\cdot\delta\mathbf{V}\,dS\right) = 0$$

Kirchhoff stress tensor

$$\int_{\Omega_0} (J\,\boldsymbol{\sigma}\cdot\mathbf{F}^{-\mathrm{T}}) : \delta\dot{\mathbf{F}}\,dV = \left(\int_{\Omega_0} \mathbf{f}^0\cdot\delta\mathbf{V}\,dV + \oint_{\Gamma_0}\mathbf{t}^0\cdot\delta\mathbf{V}\,dS\right)$$

The Principle of Virtual Displacement

$$\int_{\Omega_0} \boldsymbol{P} : \delta\dot{\mathbf{F}}\, dV - \left(\int_{\Omega_0} \mathbf{f}^0 \cdot \delta\mathbf{V}\, dV + \oint_{\Gamma_0} \mathbf{t}^0 \cdot \delta\mathbf{V}\, dS \right) = 0$$

$$\int_{\Omega_0} \boldsymbol{S} : \delta\dot{\mathbf{E}}\, dV - \left(\int_{\Omega_0} \mathbf{f}^0 \cdot \delta\mathbf{V}\, dV + \oint_{\Gamma_0} \mathbf{t}^0 \cdot \delta\mathbf{V}\, dS \right) = 0$$

Newton–Raphson Method

At time Increment n+1

$$\underbrace{\mathbf{R}}_{\text{Residual}}\left(\mathbf{d}^{\mathrm{n+1}}, t^{n+1}\right) = \mathbf{F}_{int}\left(\mathbf{d}^{\mathrm{n+1}}, t^{n+1}\right) - \mathbf{F}_{ext}\left(\mathbf{d}^{\mathrm{n+1}}, t^{n+1}\right) = 0$$

Linearized Model Of The Nonlinear Equations

At time Increment n+1
At Iteration m

$$\mathbf{R}(\mathbf{d}_{m+1}, t^{n+1}) = \mathbf{R}(\mathbf{d}_{m}, t^{n+1}) + \underbrace{\frac{\partial \mathbf{R}(\mathbf{d}_{m}, t^{n+1})}{\partial \mathbf{d}}}_{\text{Jacobian Matrix}} \underbrace{(\mathbf{d}_{\mathrm{m+1}} - \mathbf{d}_{\mathrm{m}})}_{\Delta\mathbf{d}} = 0$$

Higher Order Term Are Dropped

$$\mathbf{R}(\mathbf{d}_{m+1}, t^{n+1}) = \mathbf{0} \Rightarrow \Delta\mathbf{d} = -\left(\frac{\partial \mathbf{R}(\mathbf{d}_{m}, t^{n+1})}{\partial \mathbf{d}}\right)^{-1} \mathbf{R}(\mathbf{d}_{m}, t^{n+1}) \Rightarrow \mathbf{d}_{\mathrm{m+1}} = \mathbf{d}_{\mathrm{m}} + \Delta\mathbf{d}$$

$$\frac{\partial \mathbf{R}(\mathbf{d}_{m}, t^{n+1})}{\partial \mathbf{d}} = \frac{\partial \mathbf{F}_{int}\left(\mathbf{d}^{\mathrm{n+1}}, t^{n+1}\right)}{\partial \mathbf{d}} - \frac{\partial \mathbf{F}_{ext}\left(\mathbf{d}^{\mathrm{n+1}}, t^{n+1}\right)}{\partial \mathbf{d}}$$

$$\begin{cases} \mathbf{K}_{\mathrm{int}} = \dfrac{\partial \mathbf{F}_{int}\left(\mathbf{d}^{\mathrm{n+1}}, t^{n+1}\right)}{\partial \mathbf{d}} & \text{Tangent Stiffness Matrix} \\ \mathbf{K}_{\mathrm{ext}} = \dfrac{\partial \mathbf{F}_{ext}\left(\mathbf{d}^{\mathrm{n+1}}, t^{n+1}\right)}{\partial \mathbf{d}} & \text{Load Stiffness Matrix} \end{cases}$$

Abaqus Consistent Jacobian

For initial volume and area

$$\iiint\limits_{\Omega_0} J\boldsymbol{\sigma} : \delta\mathbf{D}\, dV = \iiint\limits_{\Omega_0} \mathbf{f}_0 \,.\, \delta\mathbf{V}\, dV + \oiint\limits_{\Gamma_0} \mathbf{t}_0 .\, \delta\mathbf{V}\, dS$$

$$\mathbf{K}_{\mathrm{int}} = \frac{\partial\mathbf{F}_{int}\left(\mathbf{d}^{\mathrm{n+1}}, t^{n+1}\right)}{\partial\mathbf{d}} \quad \Rightarrow \quad \mathbf{K}_{\mathrm{int}} = \iiint\limits_{\Omega} \frac{\partial(\boldsymbol{\sigma} : \delta\mathbf{D})}{\partial\mathbf{D}} dV$$

$$\mathbf{K}_{\mathrm{ext}} = \frac{\partial\mathbf{F}_{ext}\left(\mathbf{d}^{\mathrm{n+1}}, t^{n+1}\right)}{\partial\mathbf{d}} \quad \Rightarrow \quad \mathbf{K}_{\mathrm{ext}} = \iiint\limits_{\Omega} \frac{\partial(\mathbf{f}_0 .\, \delta\mathbf{V})}{\partial\mathbf{D}} dV + \oiint\limits_{\Gamma} \frac{\partial(\mathbf{t}_0 \,.\, \delta\mathbf{V})}{\partial\mathbf{D}} dS$$

Abaqus Consistent Jacobian

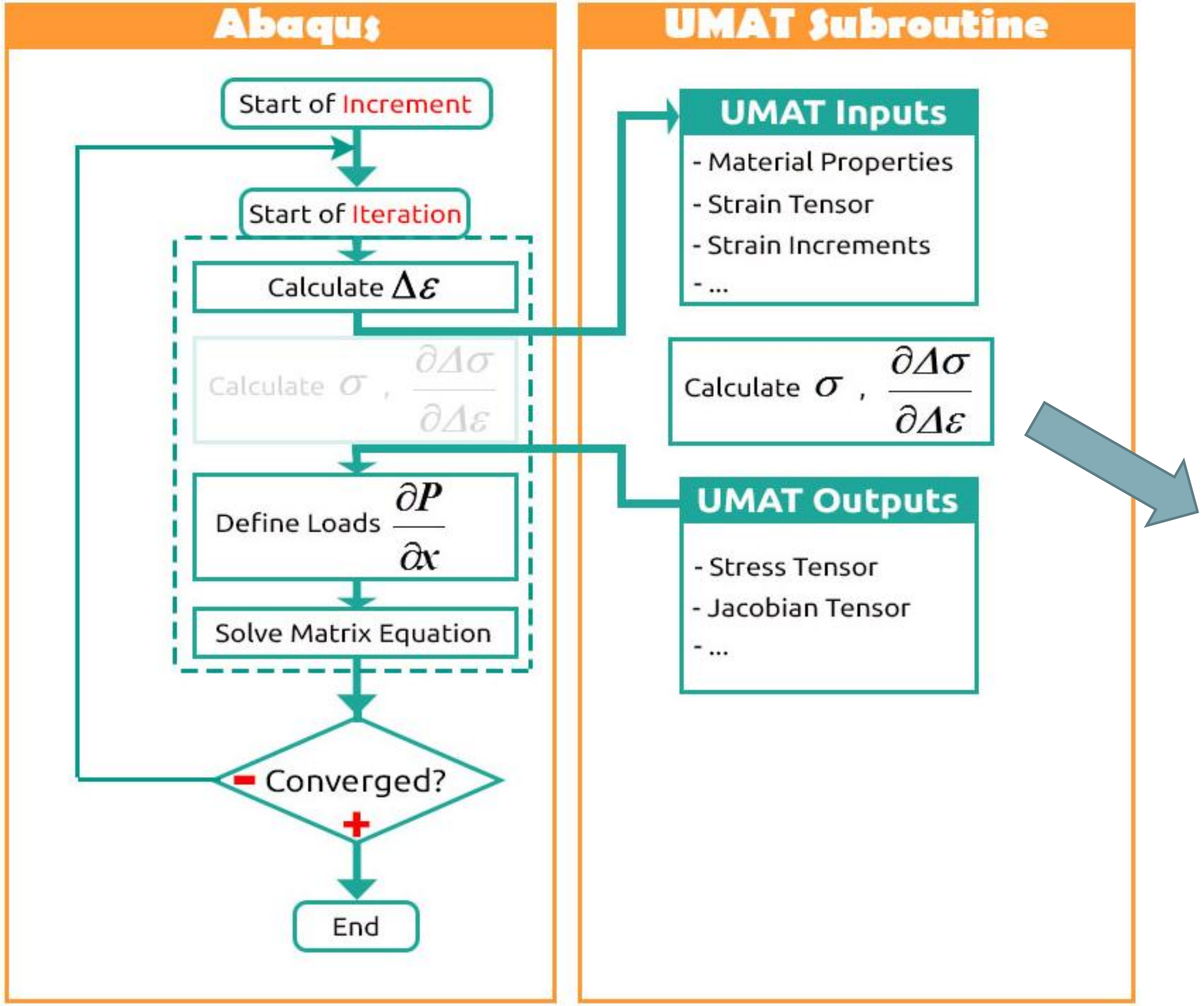
$$\mathbf{K}_{\mathrm{int}} = \iiint\limits_{\Omega} \frac{\partial\left(J\boldsymbol{\sigma} : \delta\mathbf{D}\right)}{\partial\mathbf{D}}\, dV$$

$$\mathbf{K}_{ijkl} = \iiint\limits_{\Omega_0} \frac{\partial\left(J\sigma_{ij}\ \delta D_{ij}\right)}{\partial D_{kl}}\, dV = \iiint\limits_{\Omega_0} \left[\frac{\partial\left(J\sigma_{ij}\right)}{\partial D_{kl}}\ \delta D_{ij} + \frac{\partial\left(\delta D_{ij}\right)}{\partial D_{kl}} J\sigma_{ij}\right] dV = \iiint\limits_{\Omega_0} \left[\frac{\partial\left(J\sigma_{ij}\right)}{\partial D_{kl}}\ \delta D_{ij} + \delta \mathrm{I}_{ijkl}\ (J\sigma_{ij})\right] dV$$

$$\mathbf{K}_{ijkl} = \iiint\limits_{\Omega_0} \left[\frac{\partial\left(J\sigma_{ij}\right)}{\partial D_{kl}}\ \delta D_{ij}\right] dV$$

UMAT

Abaqus User Subroutines To Define a Material's Mechanical Behavior



Abaqus

Start of Increment

Start of Iteration

Calculate $\Delta\varepsilon$

Calculate σ , $\frac{\partial\Delta\sigma}{\partial\Delta\varepsilon}$

Define Loads $\frac{\partial P}{\partial x}$

Solve Matrix Equation

Converged? − +

End

UMAT Subroutine

UMAT Inputs

- Material Properties
- Strain Tensor
- Strain Increments
- ...

Calculate σ , $\frac{\partial\Delta\sigma}{\partial\Delta\varepsilon}$

UMAT Outputs

- Stress Tensor
- Jacobian Tensor
- ...

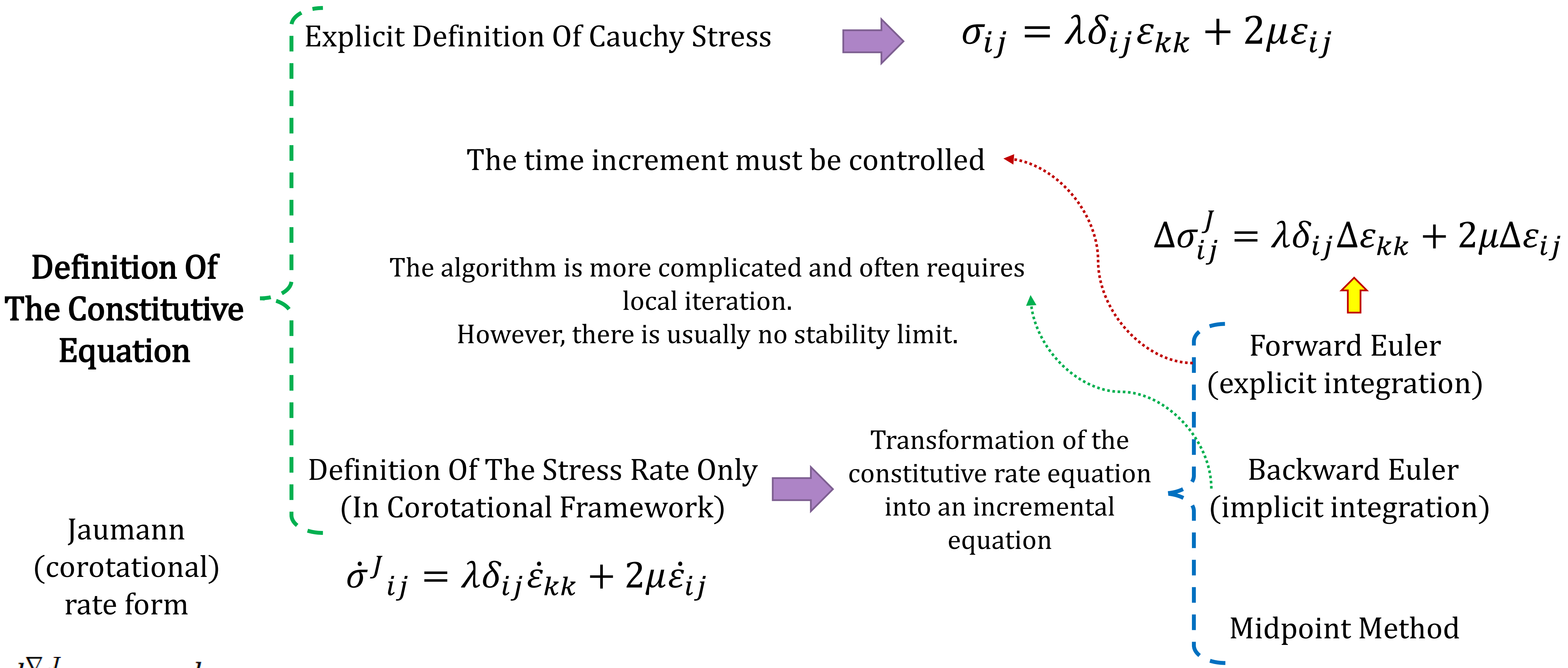
Change of stress due to rotation

$$\frac{d}{dt}(J\boldsymbol{\sigma}) = \mathbf{C}':\mathbf{D} + J(\mathbf{W}.\boldsymbol{\sigma} - \boldsymbol{\sigma}.\mathbf{W})$$

Rate of change due to material response

$$\begin{cases} \delta\left(J\boldsymbol{\sigma}\right) = J\left(\mathbf{C}:\delta\mathbf{D} + \delta\mathbf{W}\cdot\boldsymbol{\sigma} - \boldsymbol{\sigma}\cdot\delta\mathbf{W}\right) \\ \delta\mathbf{D} \overset{\text{def}}{=} \text{sym}\left(\delta\mathbf{F}\cdot\mathbf{F}^{-1}\right) \qquad \delta\mathbf{W} \overset{\text{def}}{=} \text{asym}\left(\delta\mathbf{F}\cdot\mathbf{F}^{-1}\right) \\ \mathbf{C} = \dfrac{1}{J}\dfrac{\partial\Delta\left(J\boldsymbol{\sigma}\right)}{\partial\Delta\boldsymbol{\varepsilon}}. \end{cases}$$

Isotropic Isothermal Linear Elasticity



Definition Of The Constitutive Equation

Explicit Definition Of Cauchy Stress

$$\sigma_{ij} = \lambda \delta_{ij} \varepsilon_{kk} + 2\mu \varepsilon_{ij}$$

The time increment must be controlled

$$\Delta\sigma_{ij}^{J} = \lambda \delta_{ij} \Delta\varepsilon_{kk} + 2\mu \Delta\varepsilon_{ij}$$

The algorithm is more complicated and often requires local iteration.
However, there is usually no stability limit.

Forward Euler
(explicit integration)

Backward Euler
(implicit integration)

Midpoint Method

Definition Of The Stress Rate Only
(In Corotational Framework)

Transformation of the constitutive rate equation into an incremental equation

Jaumann
(corotational)
rate form

$$\dot{\sigma}^{J}{}_{ij} = \lambda \delta_{ij} \dot{\varepsilon}_{kk} + 2\mu \dot{\varepsilon}_{ij}$$

$$\frac{d^{\nabla J}}{dt}(J\boldsymbol{\sigma}) = \frac{d}{dt}(J\boldsymbol{\sigma}) - J\left(\mathbf{W}\cdot\boldsymbol{\sigma} - \boldsymbol{\sigma}\cdot\mathbf{W}\right)$$

Isotropic Isothermal Linear Elasticity

Forward Euler
(explicit integration)

$$y(t_0+h)=y(t_0)+h\,\dot{y}(t_0) \quad\Rightarrow\quad \dot{y}(t_0)=\frac{y(t_0+h)-y(t_0)}{h}$$

$$\Delta\sigma_{ij}^{J}=\lambda\delta_{ij}\Delta\varepsilon_{kk}+2\mu\Delta\varepsilon_{ij}$$

Backward Euler
(implicit integration)

$$y(t_1-h)=y(t_1)-h\,\dot{y}(t_1) \quad\Rightarrow\quad \dot{y}(t_1)=\frac{y(t_1)-y(t_1-h=t_0)}{h}$$

Midpoint Method

$$\begin{cases} y\left(t_0+\dfrac{h}{2}\right)=y(t_0)+\dfrac{h}{2}\,\dot{y}(t_0) \\ y\left(t_0-\dfrac{h}{2}\right)=y(t_0)-\dfrac{h}{2}\,\dot{y}(t_0) \end{cases} \quad\Rightarrow\quad \dot{y}(t_0)=\frac{y\left(t_0+\frac{h}{2}\right)-y\left(t_0-\frac{h}{2}\right)}{h}$$

$$\dot{y}\left(t_0+\frac{h}{2}\right)=\frac{y(t_0+h)-y(t_0)}{h}$$

Isotropic Isothermal Linear Elasticity

Index Notation $\sigma_{ij} = \lambda\delta_{ij}\varepsilon_{kk} + 2\mu\varepsilon_{ij}$

Voigt Notation

$$\begin{bmatrix}\sigma_{11}\\\sigma_{22}\\\sigma_{33}\\\sigma_{12}\\\sigma_{13}\\\sigma_{23}\end{bmatrix} = \begin{bmatrix}2\mu+\lambda & \lambda & \lambda & 0 & 0 & 0\\ \lambda & 2\mu+\lambda & \lambda & 0 & 0 & 0\\ \lambda & \lambda & 2\mu+\lambda & 0 & 0 & 0\\ 0 & 0 & 0 & \mu & 0 & 0\\ 0 & 0 & 0 & 0 & \mu & 0\\ 0 & 0 & 0 & 0 & 0 & \mu\end{bmatrix}\begin{bmatrix}\varepsilon_{11}\\\varepsilon_{22}\\\varepsilon_{33}\\2\varepsilon_{12}\\2\varepsilon_{13}\\2\varepsilon_{23}\end{bmatrix}$$

$$\begin{bmatrix}\sigma_{11}\\\sigma_{22}\\\sigma_{33}\\\sigma_{12}\\\sigma_{13}\\\sigma_{23}\end{bmatrix} = \frac{E}{(1+\nu)(1-2\nu)}\begin{bmatrix}1-\nu & \nu & \nu & 0 & 0 & 0\\ \nu & 1-\nu & \nu & 0 & 0 & 0\\ \nu & \nu & 1-\nu & 0 & 0 & 0\\ 0 & 0 & 0 & (1-2\nu)/2 & 0 & 0\\ 0 & 0 & 0 & 0 & (1-2\nu)/2 & 0\\ 0 & 0 & 0 & 0 & 0 & (1-2\nu)/2\end{bmatrix}\begin{bmatrix}\varepsilon_{11}\\\varepsilon_{22}\\\varepsilon_{33}\\2\varepsilon_{12}\\2\varepsilon_{13}\\2\varepsilon_{23}\end{bmatrix}$$

Newton–Raphson Method

Residual

At time Increment n+1

$$\mathbf{R}\left(\mathbf{d}^{\mathrm{n+1}}, t^{n+1}\right) = \mathbf{F}_{int}\left(\mathbf{d}^{\mathrm{n+1}}, t^{n+1}\right) - \mathbf{F}_{ext}\left(\mathbf{d}^{\mathrm{n+1}}, t^{n+1}\right) = 0$$

Linearized Model Of The Nonlinear Equations

At time Increment n+1
At Iteration m

$$\mathbf{R}(\mathbf{d}_{m+1}, t^{n+1}) = \mathbf{R}(\mathbf{d}_{m}, t^{n+1}) + \underbrace{\frac{\partial \mathbf{R}(\mathbf{d}_{m}, t^{n+1})}{\partial \mathbf{d}}}_{\text{Jacobian Matrix}} \underbrace{(\mathbf{d}_{\mathrm{m+1}} - \mathbf{d}_{\mathrm{m}})}_{\Delta \mathbf{d}} = 0$$

Higher Order Term Are Dropped

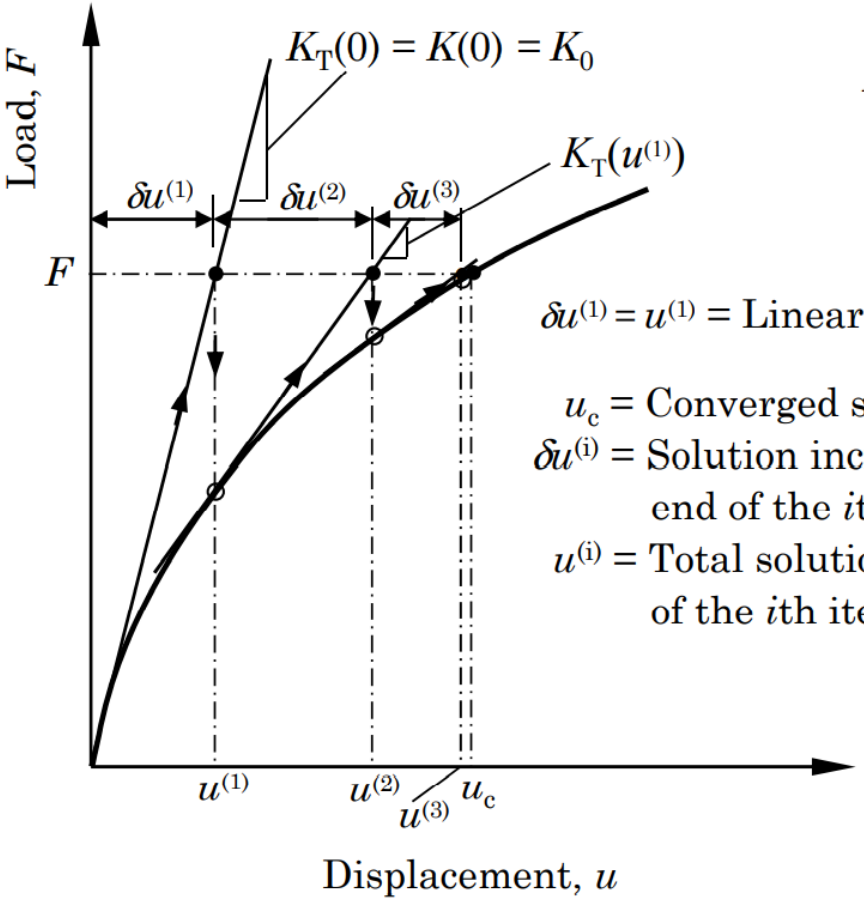
$$\mathbf{R}(\mathbf{d}_{m+1}, t^{n+1}) = \mathbf{0} \Rightarrow \Delta \mathbf{d} = -\left(\frac{\partial \mathbf{R}(\mathbf{d}_{m}, t^{n+1})}{\partial \mathbf{d}}\right)^{-1} \mathbf{R}(\mathbf{d}_{m}, t^{n+1}) \Rightarrow \mathbf{d}_{\mathrm{m+1}} = \mathbf{d}_{\mathrm{m}} + \Delta \mathbf{d}$$

$$\frac{\partial \mathbf{R}(\mathbf{d}_{m}, t^{n+1})}{\partial \mathbf{d}} = \frac{\partial \mathbf{F}_{int}\left(\mathbf{d}^{\mathrm{n+1}}, t^{n+1}\right)}{\partial \mathbf{d}} - \frac{\partial \mathbf{F}_{ext}\left(\mathbf{d}^{\mathrm{n+1}}, t^{n+1}\right)}{\partial \mathbf{d}}$$

$$\begin{cases} \mathbf{K}_{\mathrm{int}} = \dfrac{\partial \mathbf{F}_{int}\left(\mathbf{d}^{\mathrm{n+1}}, t^{n+1}\right)}{\partial \mathbf{d}} & \text{Tangent Stiffness Matrix} \\ \mathbf{K}_{\mathrm{ext}} = \dfrac{\partial \mathbf{F}_{ext}\left(\mathbf{d}^{\mathrm{n+1}}, t^{n+1}\right)}{\partial \mathbf{d}} & \text{Load Stiffness Matrix} \end{cases}$$

The Finite Element Method

$$[\mathbf{K}_e(\{\mathbf{u}_e\})]\{\mathbf{u}_e\} = \{\mathbf{F}_e\} \xrightarrow{\text{Iterative procedure}} \{\mathrm{R}\} = [\mathbf{K}_e(\{\mathbf{u}_e\})]\{\mathbf{u}_e\} - \{\mathbf{F}_e\}$$



$\delta u^{(1)} = u^{(1)}$ = Linear solution = F/K_0

u_{c} = Converged solution

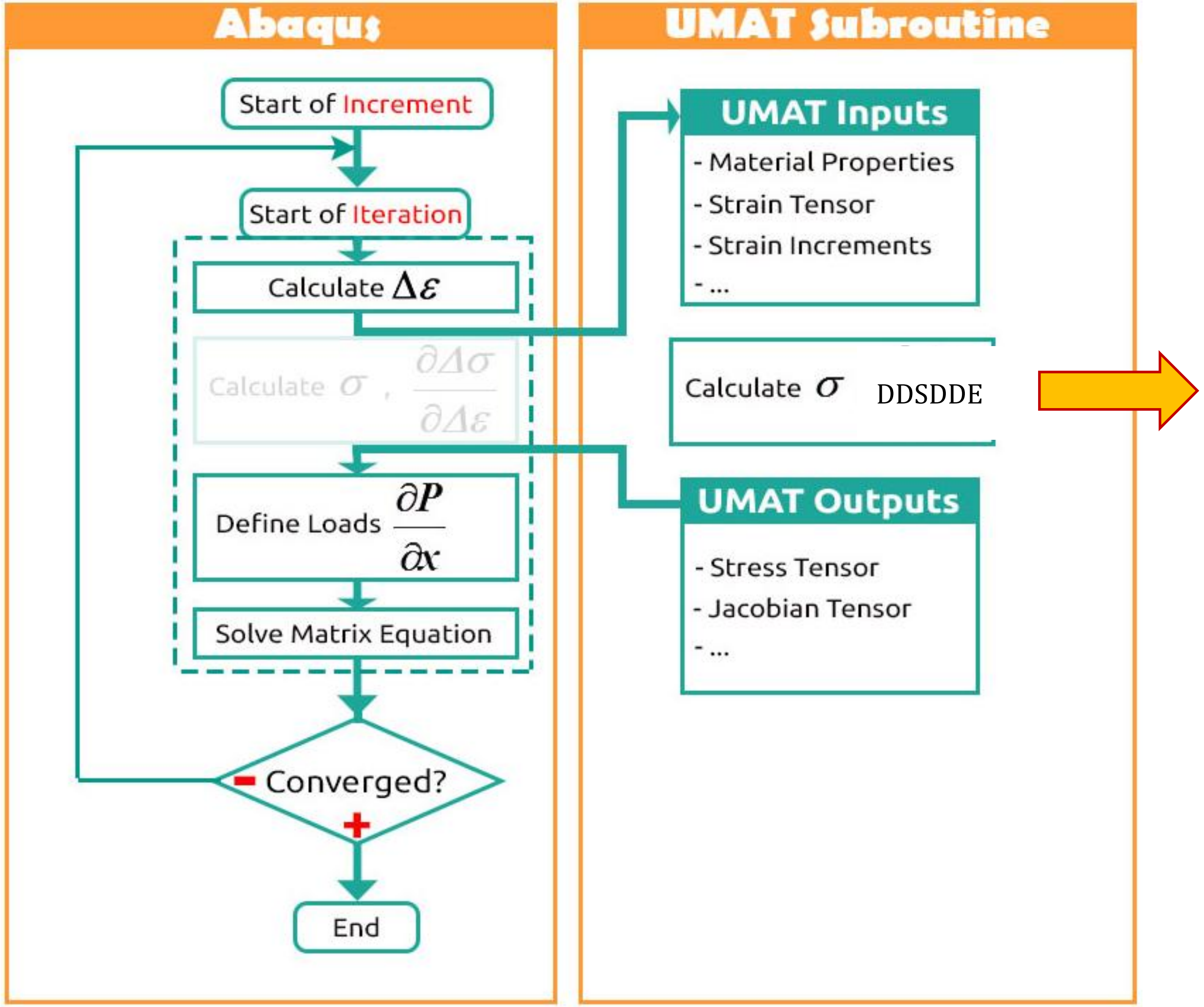
$\delta u^{(i)}$ = Solution increment at the end of the ith iteration

$u^{(i)}$ = Total solution at the end of the ith iteration (for load F)

$$R(u) = R(u^{(r-1)}) + \left(\frac{\partial R}{\partial u}\right)\bigg|_{u^{(r-1)}} \delta u + \frac{1}{2}\left(\frac{\partial^2 R}{\partial u^2}\right)\bigg|_{u^{(r-1)}} (\delta u)^2 + \cdots = 0$$

UMAT

Abaqus User Subroutines To Define a Material's Mechanical Behavior



Abaqus

Start of Increment

Start of Iteration

Calculate $\Delta\varepsilon$

Calculate σ , $\frac{\partial\Delta\sigma}{\partial\Delta\varepsilon}$

Define Loads $\frac{\partial P}{\partial x}$

Solve Matrix Equation

- Converged? +

End

UMAT Subroutine

UMAT Inputs

- Material Properties
- Strain Tensor
- Strain Increments
- ...

Calculate σ DDSDDE

UMAT Outputs

- Stress Tensor
- Jacobian Tensor
- ...

For Total-form Constitutive Laws

$$\frac{d^{\nabla J}}{dt}(J\boldsymbol{\sigma}) = \frac{d}{dt}(J\boldsymbol{\sigma}) - J(\mathbf{W}\cdot\boldsymbol{\sigma} - \boldsymbol{\sigma}\cdot\mathbf{W})$$

$$\delta(J\boldsymbol{\sigma}) = J(\mathbf{C}:\delta\mathbf{D} + \delta\mathbf{W}\cdot\boldsymbol{\sigma} - \boldsymbol{\sigma}\cdot\delta\mathbf{W})$$

$$\delta\mathbf{D} \overset{\text{def}}{=} \text{sym}\left(\delta\mathbf{F}\cdot\mathbf{F}^{-1}\right)$$

For Rate-form Constitutive Laws

$$\mathbf{C} = \frac{1}{J}\frac{\partial\Delta(J\boldsymbol{\sigma})}{\partial\Delta\boldsymbol{\varepsilon}}.$$

Isotropic Non-isothermal Linear Elasticity

Definition Of The Constitutive Equation

Explicit Definition Of Cauchy Stress

$$\sigma_{ij} = \lambda(T)\delta_{ij}\varepsilon_{kk}^{el} + 2\mu(T)\varepsilon_{ij}^{el} \qquad \varepsilon_{ij}^{el} = \varepsilon_{ij} - \alpha T\delta_{ij}$$

Definition Of The Stress Rate Only (In Corotational Framework)

$$\dot{\sigma}_{ij}^{J} = \lambda\delta_{ij}\dot{\varepsilon}_{kk}^{el} + 2\mu\dot{\varepsilon}_{ij}^{el} + \dot{\lambda}\delta_{ij}\varepsilon_{kk}^{el} + 2\dot{\mu}\varepsilon_{ij}^{el}$$

$$\dot{\varepsilon}_{ij}^{el} = \dot{\varepsilon}_{ij} - \alpha\dot{T}\delta_{ij}$$

Transformation of the constitutive rate equation into an incremental equation

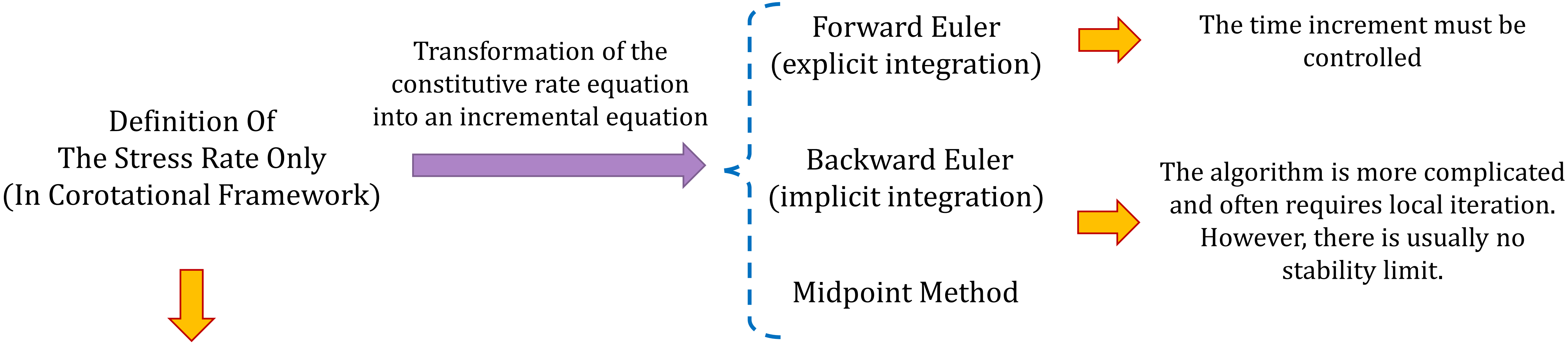
$$\Delta\sigma_{ij}^{J} = \lambda\delta_{ij}\Delta\varepsilon_{kk}^{el} + 2\mu\Delta\varepsilon_{ij}^{el} + \Delta\lambda\delta_{ij}\varepsilon_{kk}^{el} + 2\Delta\mu\varepsilon_{ij}^{el}$$

$$\Delta\varepsilon_{ij}^{el} = \Delta\varepsilon_{ij} - \alpha\Delta T\delta_{ij}$$

Jaumann (corotational) rate form

$$\frac{d^{\nabla J}}{dt}(J\boldsymbol{\sigma}) = \frac{d}{dt}(J\boldsymbol{\sigma}) - J(\mathbf{W}\cdot\boldsymbol{\sigma} - \boldsymbol{\sigma}\cdot\mathbf{W})$$

Isotropic Non-isothermal Linear Elasticity



Definition Of
The Stress Rate Only
(In Corotational Framework)

Transformation of the constitutive rate equation into an incremental equation

Forward Euler
(explicit integration)

The time increment must be controlled

Backward Euler
(implicit integration)

The algorithm is more complicated and often requires local iteration. However, there is usually no stability limit.

Midpoint Method

Jaumann (corotational) rate form

$$\frac{d^{\nabla J}}{dt}(J\boldsymbol{\sigma}) = \frac{d}{dt}(J\boldsymbol{\sigma}) - J\left(\mathbf{W}\cdot\boldsymbol{\sigma} - \boldsymbol{\sigma}\cdot\mathbf{W}\right)$$

Isotropic Non-isothermal Linear Elasticity

Forward Euler
(explicit integration) $\Rightarrow$ $y(t_0+h)=y(t_0)+h\,\dot{y}(t_0)$ $\Rightarrow$ $\dot{y}(t_0)=\dfrac{y(t_0+h)-y(t_0)}{h}$

$$\Delta\sigma_{ij}^{J}=\lambda\delta_{ij}\Delta\varepsilon_{kk}^{el}+2\mu\Delta\varepsilon_{ij}^{el}+\Delta\lambda\delta_{ij}\varepsilon_{kk}^{el}+2\Delta\mu\varepsilon_{ij}$$

$$\Delta\varepsilon_{ij}^{el}=\Delta\varepsilon_{ij}-\alpha\Delta T\delta_{ij}$$

Backward Euler
(implicit integration) $\Rightarrow$ $y(t_1-h)=y(t_1)-h\,\dot{y}(t_1)$ $\Rightarrow$ $\dot{y}(t_1)=\dfrac{y(t_1)-y(t_1-h=t_0)}{h}$

Midpoint Method $\Rightarrow$

$$\begin{cases} y\left(t_0+\dfrac{h}{2}\right)=y(t_0)+\dfrac{h}{2}\,\dot{y}(t_0) \\ y\left(t_0-\dfrac{h}{2}\right)=y(t_0)-\dfrac{h}{2}\,\dot{y}(t_0) \end{cases}$$

$\Rightarrow$ $\dot{y}(t_0)=\dfrac{y\left(t_0+\dfrac{h}{2}\right)-y\left(t_0-\dfrac{h}{2}\right)}{h}$

$\Rightarrow$ $\dot{y}\left(t_0+\dfrac{h}{2}\right)=\dfrac{y(t_0+h)-y(t_0)}{h}$

Isotropic Non-isothermal Linear Elasticity

Index Notation $\Delta\sigma_{ij}^{J} = \lambda\delta_{ij}\Delta\varepsilon_{kk}^{el} + 2\mu\Delta\varepsilon_{ij}^{el} + \Delta\lambda\delta_{ij}\varepsilon_{kk}^{el} + 2\Delta\mu\varepsilon_{ij}^{el}$ $\Delta\varepsilon_{ij}^{el} = \Delta\varepsilon_{ij} - \alpha\Delta T\delta_{ij}$

Voigt Notation

$$\begin{bmatrix}\sigma_{11}\\\sigma_{22}\\\sigma_{33}\\\sigma_{12}\\\sigma_{13}\\\sigma_{23}\end{bmatrix} = \begin{bmatrix}2\mu+\lambda & \lambda & \lambda & 0 & 0 & 0\\ \lambda & 2\mu+\lambda & \lambda & 0 & 0 & 0\\ \lambda & \lambda & 2\mu+\lambda & 0 & 0 & 0\\ 0 & 0 & 0 & \mu & 0 & 0\\ 0 & 0 & 0 & 0 & \mu & 0\\ 0 & 0 & 0 & 0 & 0 & \mu\end{bmatrix}\begin{bmatrix}\varepsilon_{11}\\\varepsilon_{22}\\\varepsilon_{33}\\2\varepsilon_{12}\\2\varepsilon_{13}\\2\varepsilon_{23}\end{bmatrix}$$

$$\begin{bmatrix}\sigma_{11}\\\sigma_{22}\\\sigma_{33}\\\sigma_{12}\\\sigma_{13}\\\sigma_{23}\end{bmatrix} = \frac{E}{(1+\nu)(1-2\nu)}\begin{bmatrix}1-\nu & \nu & \nu & 0 & 0 & 0\\ \nu & 1-\nu & \nu & 0 & 0 & 0\\ \nu & \nu & 1-\nu & 0 & 0 & 0\\ 0 & 0 & 0 & (1-2\nu)/2 & 0 & 0\\ 0 & 0 & 0 & 0 & (1-2\nu)/2 & 0\\ 0 & 0 & 0 & 0 & 0 & (1-2\nu)/2\end{bmatrix}\begin{bmatrix}\varepsilon_{11}\\\varepsilon_{22}\\\varepsilon_{33}\\2\varepsilon_{12}\\2\varepsilon_{13}\\2\varepsilon_{23}\end{bmatrix}$$

Linear Interpolation

$$E(T) = N_1 E(T_1) + N_2 E(T_2)$$

$$\nu(T) = N_1 \nu(T_1) + N_2 \nu(T_2)$$

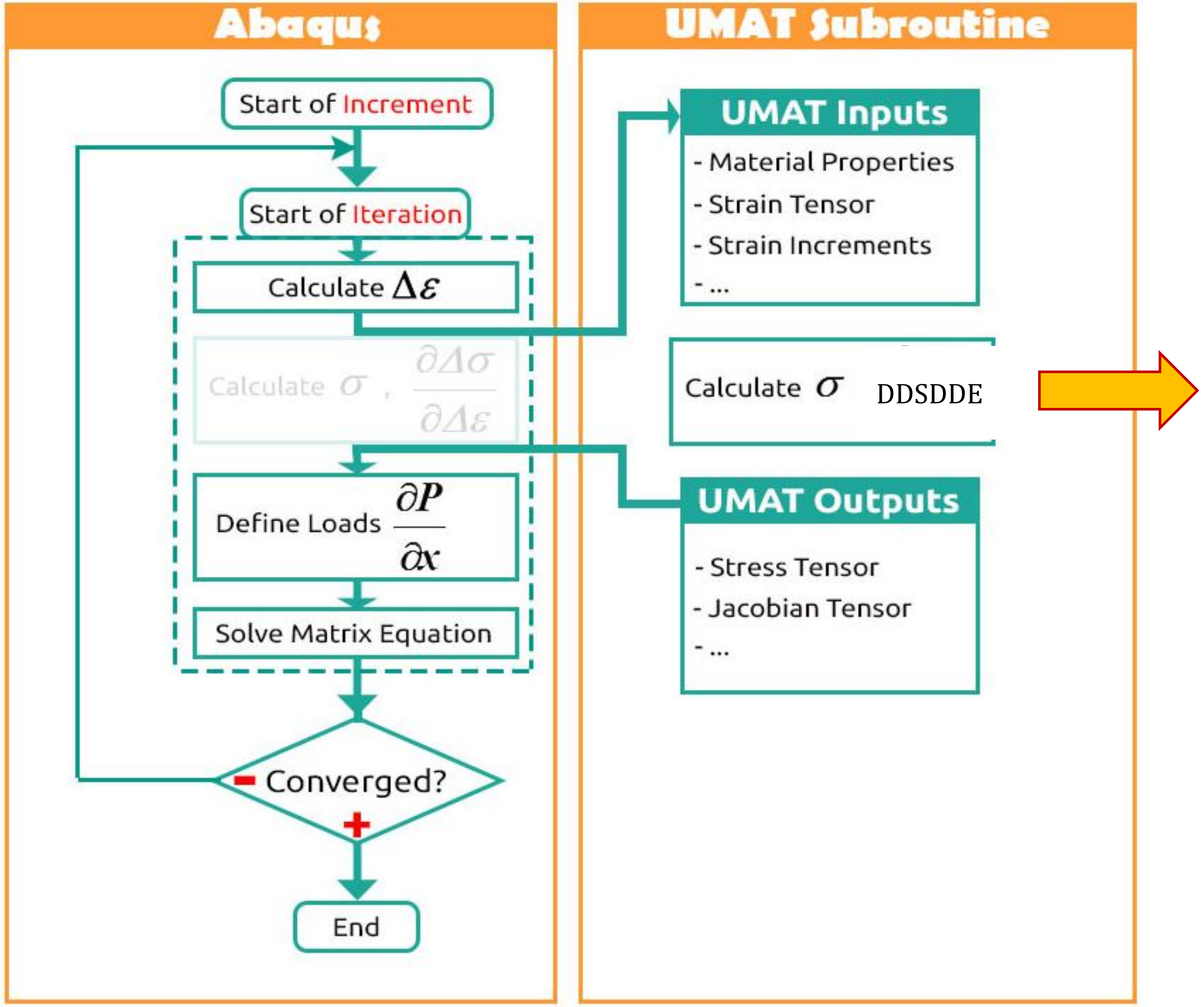
$$N_1 = \frac{T_2 - T}{T_2 - T_1} \qquad N_2 = \frac{T - T_1}{T_2 - T_1} \qquad N_1 + N_2 = 1$$

$$E(T) - E(T_1) = \frac{E(T_2) - E(T_1)}{T_2 - T_1}(T - T_1) \Longrightarrow E(T) = \frac{T - T_1}{T_2 - T_1} E(T_2) - \frac{T - T_1}{T_2 - T_1} E(T_1) + E(T_1)$$

$$E(T) = \underbrace{\frac{T_2 - T}{T_2 - T_1}}_{N_1} E(T_1) + \underbrace{\frac{T - T_1}{T_2 - T_1}}_{N_2} E(T_2)$$

UMAT

Abaqus User Subroutines To Define a Material's Mechanical Behavior



For Total-form Constitutive Laws

$$\frac{d^{\nabla J}}{dt}\left(J\boldsymbol{\sigma}\right) = \frac{d}{dt}\left(J\boldsymbol{\sigma}\right) - J\left(\mathbf{W}\cdot\boldsymbol{\sigma} - \boldsymbol{\sigma}\cdot\mathbf{W}\right)$$

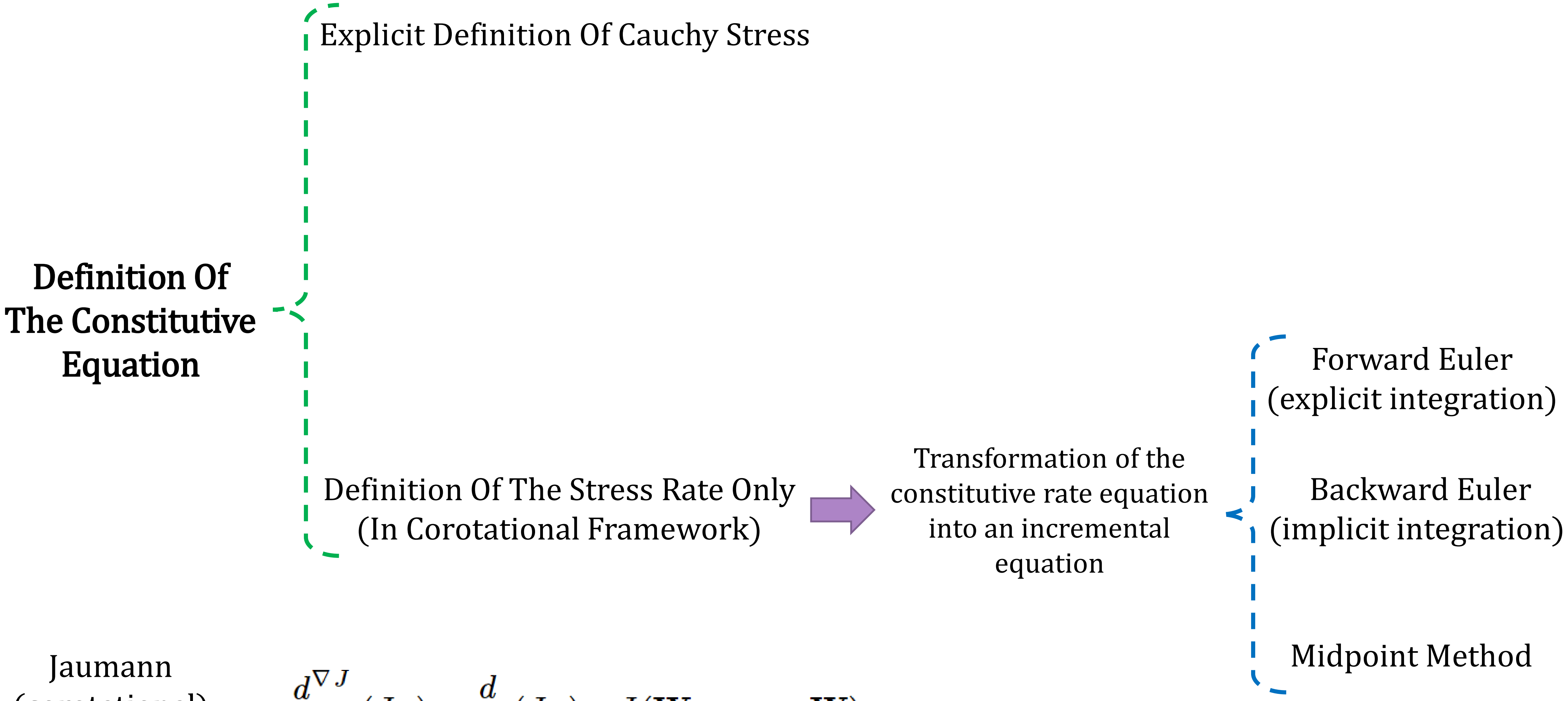
$$\delta\left(J\boldsymbol{\sigma}\right) = J\left(\mathbf{C} : \delta\mathbf{D} + \delta\mathbf{W}\cdot\boldsymbol{\sigma} - \boldsymbol{\sigma}\cdot\delta\mathbf{W}\right)$$

$$\delta\mathbf{D} \overset{\text{def}}{=} \text{sym}\left(\delta\mathbf{F}\cdot\mathbf{F}^{-1}\right)$$

For Rate-form Constitutive Laws

$$\mathbf{C} = \frac{1}{J}\frac{\partial\,\Delta\left(J\boldsymbol{\sigma}\right)}{\partial\,\Delta\boldsymbol{\varepsilon}}.$$

Green Elastic Material (Hyperelastic Material)



Definition Of The Constitutive Equation

Explicit Definition Of Cauchy Stress

Definition Of The Stress Rate Only (In Corotational Framework)

Transformation of the constitutive rate equation into an incremental equation

Forward Euler (explicit integration)

Backward Euler (implicit integration)

Midpoint Method

Jaumann (corotational) rate form

$$\frac{d^{\nabla J}}{dt}(J\boldsymbol{\sigma}) = \frac{d}{dt}(J\boldsymbol{\sigma}) - J\left(\mathbf{W}\cdot\boldsymbol{\sigma} - \boldsymbol{\sigma}\cdot\mathbf{W}\right)$$

Green Elastic Material (Hyperelastic Material)

Volume-preserving, Or Isochoric Part of **F**

Deformation Gradient

$$\mathbf{F} = \nabla_0 \mathbf{x} = \frac{\partial \mathbf{x}\,(\mathbf{X}, t)}{\partial \mathbf{X}}$$

Distortion Gradient

$$\bar{\mathbf{F}} = J^{-\frac{1}{3}}\,\mathbf{F}$$

Jacobian Determinant

$$\bar{\mathbf{C}} = \bar{\mathbf{F}}^T.\bar{\mathbf{F}}$$

Deviatoric Right Cauchy-green Deformation Tensor

$$\bar{\mathbf{B}} = \bar{\mathbf{F}}.\bar{\mathbf{F}}^T$$

Deviatoric Left Cauchy-green Deformation Tensor

Compressible Mooney–Rivlin Hyperelasticity

$$U\left(\bar{I}_1, \bar{I}_2, I_3 = \sqrt{J^{el}}\right) = C_{10}(\bar{I}_1 - 3) + C_{01}(\bar{I}_2 - 3) + \frac{1}{D_1}\left(J^{el} - 1\right)^2$$

$$J^{el} = \frac{J}{J^{th}}$$

$$\left\{\begin{array}{l} \bar{I}_1 = \left(\bar{\lambda}_1\right)^2 + \left(\bar{\lambda}_2\right)^2 + \left(\bar{\lambda}_3\right)^2 = tr(\bar{B}) = tr(\bar{C}) \\ \bar{I}_2 = \left(\bar{\lambda}_1\right)^{-2} + \left(\bar{\lambda}_2\right)^{-2} + \left(\bar{\lambda}_3\right)^{-2} = \frac{1}{2}\left(tr(\bar{B})^2 - tr(\,\bar{B}.\bar{B})\right) = \frac{1}{2}\left(tr(\bar{C})^2 - tr(\,\bar{C}.\bar{C})\right) \\ I_3 = \sqrt{J^{el}} \end{array}\right.$$

- $\bar{I}_i$:Deviatoric Invariants
- $\bar{\lambda}_i$: Deviatoric Stretches
- J^{el}: Elastic Volume Ratio
- J: Total Volume Ratio

Compressible Mooney–Rivlin Hyperelasticity

$$U\left(\bar{I}_1,\bar{I}_2,I_3=\sqrt{J^{el}}\right)=C_{10}(\bar{I}_1-3)+C_{01}(\bar{I}_2-3)+\frac{1}{D_1}\left(J^{el}-1\right)^2 \qquad J^{el}=\frac{J}{J^{th}}$$

$\bar{I}_i$:Deviatoric Invariants

$\bar{\lambda}_i$: Deviatoric Stretches

J^{el}: Elastic Volume Ratio

J: Total Volume Ratio

$$\begin{cases}\bar{I}_1=\left(\bar{\lambda}_1\right)^2+\left(\bar{\lambda}_2\right)^2+\left(\bar{\lambda}_3\right)^2=tr(\bar{B})=tr(\bar{C})\\ \bar{I}_2=\left(\bar{\lambda}_1\right)^{-2}+\left(\bar{\lambda}_2\right)^{-2}+\left(\bar{\lambda}_3\right)^{-2}=\frac{1}{2}\left(tr(\bar{B})^2-tr(\,\bar{B}.\bar{B})\right)=\frac{1}{2}\left(tr(\bar{C})^2-tr(\,\bar{C}.\bar{C})\right)\\ I_3=\sqrt{J^{el}}\end{cases}$$

$$\mathbf{S}=2\frac{\partial U}{\partial \mathbf{C}}=2\left[\frac{\partial U}{\partial \bar{I}_1}\frac{\partial \bar{I}_1}{\partial \mathbf{C}}+\frac{\partial U}{\partial \bar{I}_2}\frac{\partial \bar{I}_2}{\partial \mathbf{C}}+\frac{\partial U}{\partial J^{el}}\frac{\partial J^{el}}{\partial \mathbf{C}}\right] \Rightarrow \boldsymbol{\sigma}=\frac{1}{J}\mathbf{F}.\mathbf{S}.\mathbf{F}^T$$

$$\sigma_{ij}=\frac{2}{J}C_{10}\left(\bar{B}_{ij}-\frac{1}{3}\delta_{ij}\bar{B}_{kk}\right)+\frac{2}{J}C_{01}\left(\bar{B}_{kk}\bar{B}_{ij}-\frac{1}{3}\delta_{ij}\,(\bar{B}_{kk})^2-\bar{B}_{ik}\bar{B}_{kj}+\frac{1}{3}\delta_{ij}\bar{B}_{kn}\bar{B}_{nk}\right)+\frac{2}{D_1}(J^{el}-1)\delta_{ij}$$

Compressible Mooney–Rivlin Hyperelasticity

$$\sigma_{ij} = \frac{2}{J} C_{10} \left(\bar{B}_{ij} - \frac{1}{3} \delta_{ij} \bar{B}_{kk} \right) + \frac{2}{J} C_{01} \left(\bar{B}_{kk} \bar{B}_{ij} - \frac{1}{3} \delta_{ij} \left(\bar{B}_{kk} \right)^2 - \bar{B}_{ik} \bar{B}_{kj} + \frac{1}{3} \delta_{ij} \bar{B}_{kn} \bar{B}_{nk} \right) + \frac{2}{D_1} \left(J^{el} - 1 \right) \delta_{ij}$$

$$\delta \left(J \boldsymbol{\sigma} \right) = J \left(\mathbf{C} : \delta \mathbf{D} + \delta \mathbf{W} \cdot \boldsymbol{\sigma} - \boldsymbol{\sigma} \cdot \delta \mathbf{W} \right)$$

$$\delta \left(J \sigma_{ij} \right) - J \left(\delta W_{ik} \sigma_{kj} + \sigma_{ij} \delta W_{kj} \right) = J C_{ijkl} \delta D_{kl} \quad \Longrightarrow \quad \begin{cases} \delta D_{ij} = \dfrac{1}{2} \left(\delta F_{im} F_{mj}^{-1} + F_{mi}^{-1} \delta F_{jm} \right) \\ \delta W_{ij} = \dfrac{1}{2} \left(\delta F_{im} F_{mj}^{-1} - F_{mi}^{-1} \delta F_{jm} \right) \end{cases}$$

$$C_{ijkl} = \frac{2}{J} C_{10} \left[\frac{1}{2} \left(\delta_{ik} \bar{B}_{jl} + \bar{B}_{ik} \delta_{jl} + \delta_{il} \bar{B}_{jk} + \bar{B}_{il} \delta_{jk} \right) - \frac{2}{3} \delta_{ij} \bar{B}_{kl} - \frac{2}{3} \bar{B}_{ij} \delta_{kl} + \frac{2}{9} \delta_{ij} \delta_{kl} \bar{B}_{mm} \right] + \frac{2}{D_1} (2J - 1) \delta_{ij} \delta_{kl}$$

$$\begin{cases} \mathbf{C}^e = 4 \mathbf{B} \, . \, \dfrac{\partial^2 U}{\partial \mathbf{B} \otimes \partial \mathbf{B}} \, . \, \mathbf{B} \\ \mathbf{C} = \dfrac{1}{J} \mathbf{C}^e + \dfrac{1}{2} \left[\boldsymbol{\sigma} \,\bar{\otimes}\, \mathbf{I} + \mathbf{I} \,\bar{\otimes}\, \boldsymbol{\sigma} + \boldsymbol{\sigma} \,\underline{\otimes}\, \mathbf{I} + \mathbf{I} \,\underline{\otimes}\, \boldsymbol{\sigma} \right] \end{cases}$$

$$C^e_{ijkl} = 4 B_{im} \frac{\partial^2 U}{\partial B_{mj} \, \partial B_{kn}} B_{nl}$$

$$\{ * \otimes \circ \}_{ijkl} = \{ * \}_{ij} \{ \circ \}_{kl}$$

$$\{ * \bar{\otimes} \circ \}_{ijkl} = \{ * \}_{ik} \{ \circ \}_{jl}$$

$$\{ * \underline{\otimes} \circ \}_{ijkl} = \{ * \}_{il} \{ \circ \}_{jk}$$

Compressible Mooney–Rivlin Hyperelasticity

The convention used for stress and strain components in Abaqus is that they are ordered:

$$\begin{pmatrix}\sigma_{11}\\ \sigma_{22}\\ \sigma_{33}\\ \tau_{12}\\ \tau_{13}\\ \tau_{23}\end{pmatrix} = \begin{bmatrix} D_{1111} & D_{1122} & D_{1133} & D_{1112} & D_{1113} & D_{1123}\\ & D_{2222} & D_{2233} & D_{2212} & D_{2213} & D_{2223}\\ & & D_{3333} & D_{3312} & D_{3313} & D_{3323}\\ & \text{symm.} & & D_{1212} & D_{1213} & D_{1223}\\ & & & & D_{1313} & D_{1323}\\ & & & & & D_{2323}\end{bmatrix} \begin{pmatrix}\varepsilon_{11}\\ \varepsilon_{22}\\ \varepsilon_{33}\\ \gamma_{12}\\ \gamma_{13}\\ \gamma_{23}\end{pmatrix}$$

σ_{11}	Direct stress in the 1-direction
σ_{22}	Direct stress in the 2-direction
σ_{33}	Direct stress in the 3-direction
τ_{12}	Shear stress in the 1–2 plane
τ_{13}	Shear stress in the 1–3 plane
τ_{23}	Shear stress in the 2–3 plane

Compressible Neo-Hookean Hyperelasticity

$$\psi = \frac{1}{2}\lambda\big(ln(J_e)\big)^2 + \frac{1}{2}\mu[I_1 - 3 - 2ln(J_e)]$$

$$\lambda = \frac{E\nu}{(1+\nu)(1-2\nu)} \qquad \mu = \frac{E}{2(1+\nu)}$$

$$\{*\otimes\circ\}_{ijkl} = \{*\}_{ij}\,\{\circ\}_{kl}$$

$$\{*\bar{\otimes}\circ\}_{ijkl} = \{*\}_{ik}\,\{\circ\}_{jl}$$

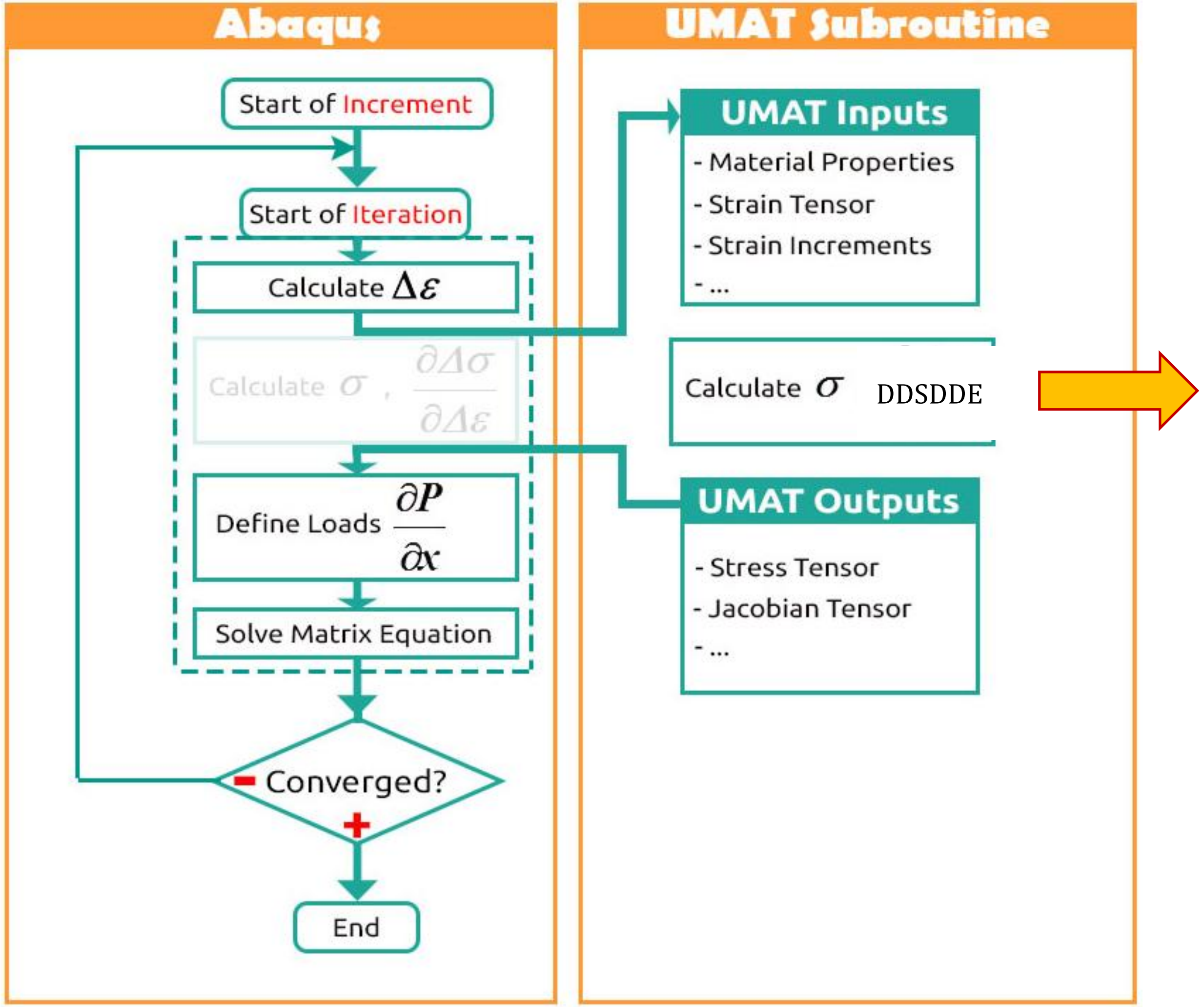
$$\{*\underline{\otimes}\circ\}_{ijkl} = \{*\}_{il}\,\{\circ\}_{jk}$$

$$\boldsymbol{\tau} = 2\frac{\partial\psi}{\partial\mathbf{B}}.\mathbf{B} = \mathbf{F.S.F}^T = [\lambda\; ln(J_e) - \mu]\,\mathbf{I} + \mu\,\mathbf{B}$$

$$\mathbf{C} = \frac{1}{J}\Big\{\lambda\,\mathbf{I}\otimes\mathbf{I} + [\mu - \lambda\; ln(J_e)]\big[\mathbf{I}\,\bar{\otimes}\,\mathbf{I} + \mathbf{I}\,\underline{\otimes}\,\mathbf{I}\big] + \frac{1}{2}\big[\boldsymbol{\tau}\,\bar{\otimes}\,\mathbf{I} + \mathbf{I}\,\bar{\otimes}\,\boldsymbol{\tau} + \boldsymbol{\tau}\,\underline{\otimes}\,\mathbf{I} + \mathbf{I}\,\underline{\otimes}\,\boldsymbol{\tau}\big]\Big\}$$

UMAT

Abaqus User Subroutines To Define a Material's Mechanical Behavior



Abaqus

Start of Increment

Start of Iteration

Calculate $\Delta\varepsilon$

Calculate σ , $\frac{\partial\Delta\sigma}{\partial\Delta\varepsilon}$

Define Loads $\frac{\partial P}{\partial x}$

Solve Matrix Equation

Converged?

−

+

End

UMAT Subroutine

UMAT Inputs

- Material Properties
- Strain Tensor
- Strain Increments
- ...

Calculate σ DDSDDE

UMAT Outputs

- Stress Tensor
- Jacobian Tensor
- ...

For Total-form Constitutive Laws

$$\frac{d^{\nabla J}}{dt}\left(J\boldsymbol{\sigma}\right)=\frac{d}{dt}\left(J\boldsymbol{\sigma}\right)-J\left(\mathbf{W}\cdot\boldsymbol{\sigma}-\boldsymbol{\sigma}\cdot\mathbf{W}\right)$$

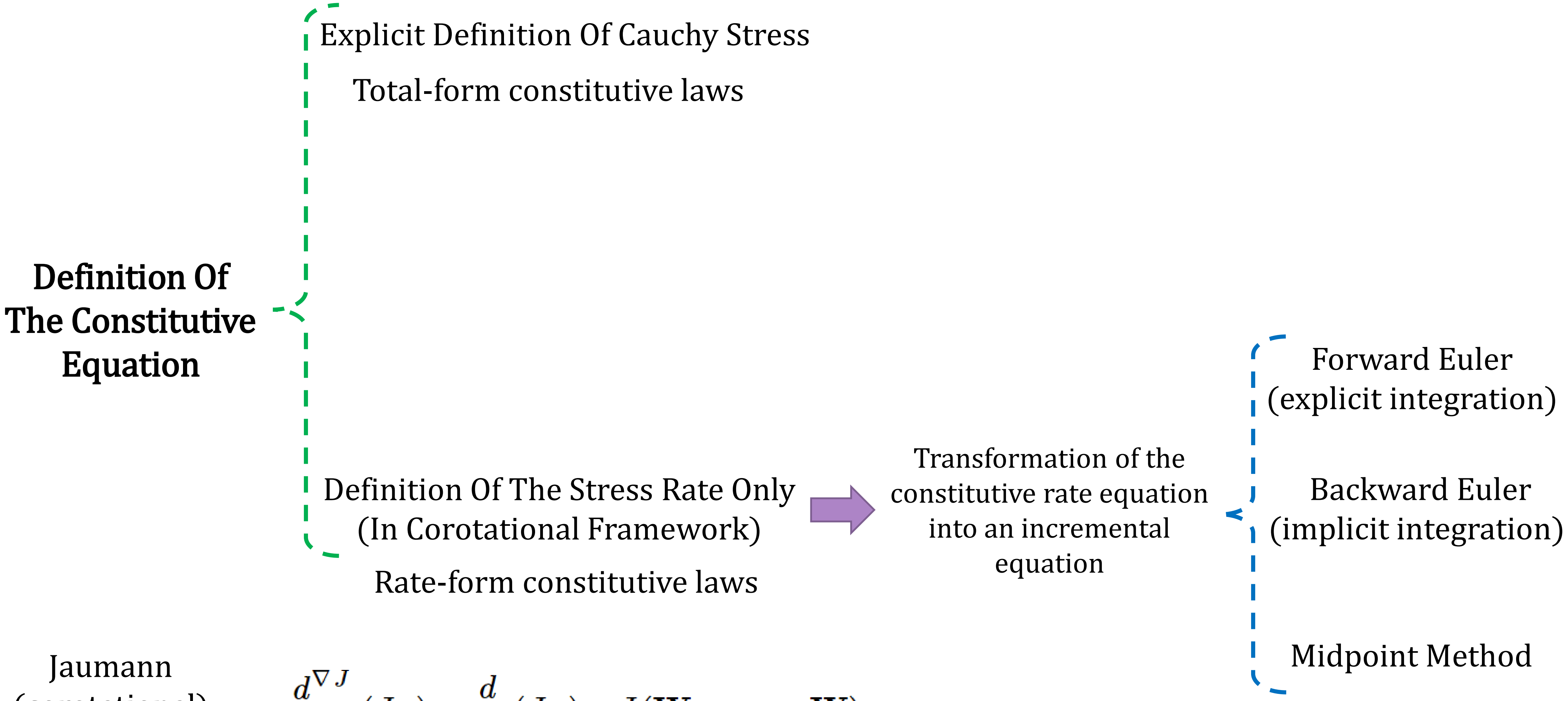
$$\delta\left(J\boldsymbol{\sigma}\right)=J\left(\mathbf{C}:\delta\mathbf{D}+\delta\mathbf{W}\cdot\boldsymbol{\sigma}-\boldsymbol{\sigma}\cdot\delta\mathbf{W}\right)$$

$$\delta\mathbf{D}\overset{\text{def}}{=}\text{sym}\left(\delta\mathbf{F}\cdot\mathbf{F}^{-1}\right)\qquad\delta\mathbf{W}\overset{\text{def}}{=}\text{asym}\left(\delta\mathbf{F}\cdot\mathbf{F}^{-1}\right)$$

For Rate-form Constitutive Laws

$$\mathbf{C}=\frac{1}{J}\frac{\partial\Delta\left(J\boldsymbol{\sigma}\right)}{\partial\Delta\boldsymbol{\varepsilon}}.$$

Green Elastic Material (Hyperelastic Material)



Definition Of The Constitutive Equation

Explicit Definition Of Cauchy Stress

Total-form constitutive laws

Definition Of The Stress Rate Only (In Corotational Framework)

Rate-form constitutive laws

Transformation of the constitutive rate equation into an incremental equation

Forward Euler (explicit integration)

Backward Euler (implicit integration)

Midpoint Method

Jaumann (corotational) rate form

$$\frac{d^{\nabla J}}{dt}(J\boldsymbol{\sigma}) = \frac{d}{dt}(J\boldsymbol{\sigma}) - J\left(\mathbf{W}\cdot\boldsymbol{\sigma} - \boldsymbol{\sigma}\cdot\mathbf{W}\right)$$

Almost Incompressible or Fully Incompressible Elastic Materials

few different options are available depending on whether **hybrid** or **nonhybrid** elements are used

Option 1 → For all cases the first option should be to use user subroutine **UHYPER** instead of user subroutine UMAT when it is possible to do so

Option 2 → In user subroutine UMAT incompressible materials can be modeled via a penalty method; that is, you ensure that a finite bulk modulus is used.

Almost Incompressible

The bulk modulus should be large enough to model incompressibility sufficiently but small enough to avoid loss of precision

As a general guideline, the bulk modulus should be about $10^4 - 10^6$ times the shear modulus

$$K = -V\frac{dP}{dV}$$

The tangent bulk modulus

$$K^t = \frac{1}{9}\sum_{\mathrm{I}=1}^{3}\sum_{\mathrm{J}=1}^{3}\mathrm{DDSDDE}\,(\mathrm{I},\mathrm{J})$$

Almost Incompressible or Fully Incompressible Elastic Materials

few different options are available depending on whether **hybrid** or **nonhybrid** elements are used

Option 1

For all cases the first option should be to use user subroutine **UHYPER** instead of user subroutine UMAT when it is possible to do so

Option 2

In user subroutine UMAT incompressible materials can be modeled via a penalty method; that is, you ensure that a finite bulk modulus is used.

Hybrid Element

Abaqus/Standard, by default, replaces the pressure stress calculated from your definition of STRESS with that derived from the Lagrange multiplier and modifies the Jacobian appropriately

Suitable for material models that use an **incremental** formulation (metal plasticity)

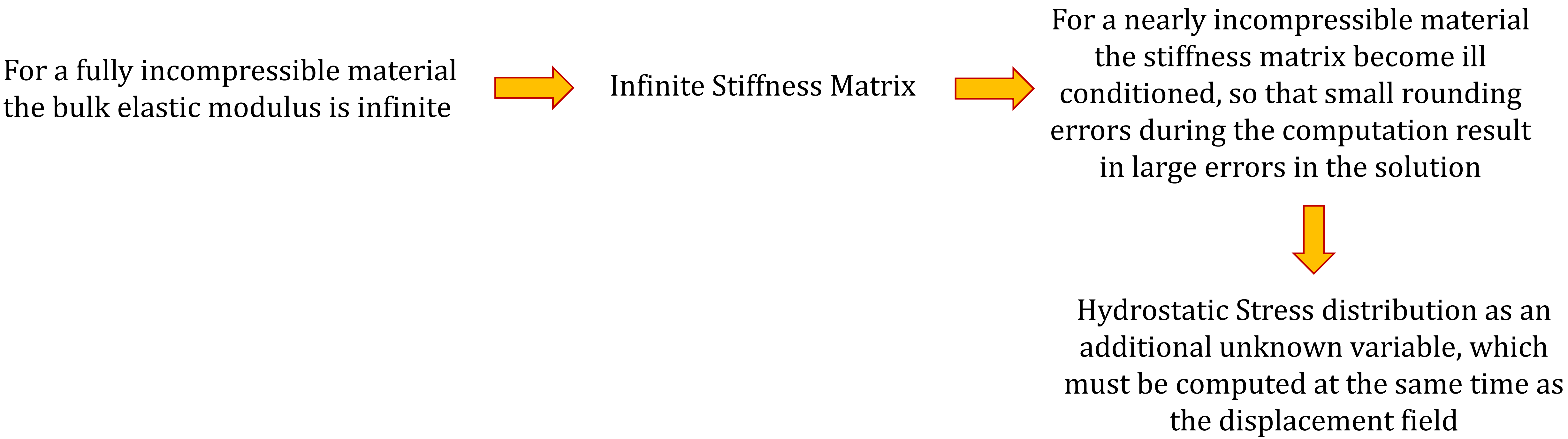
but is not consistent with a total formulation that is commonly used for hyperelastic materials

lead to convergence problems

Nonhybrid Element

Hybrid Elements

Hybrid Elements are used to Modeling Near-Incompressible and Fully incompressible Materials



For a fully incompressible material the bulk elastic modulus is infinite

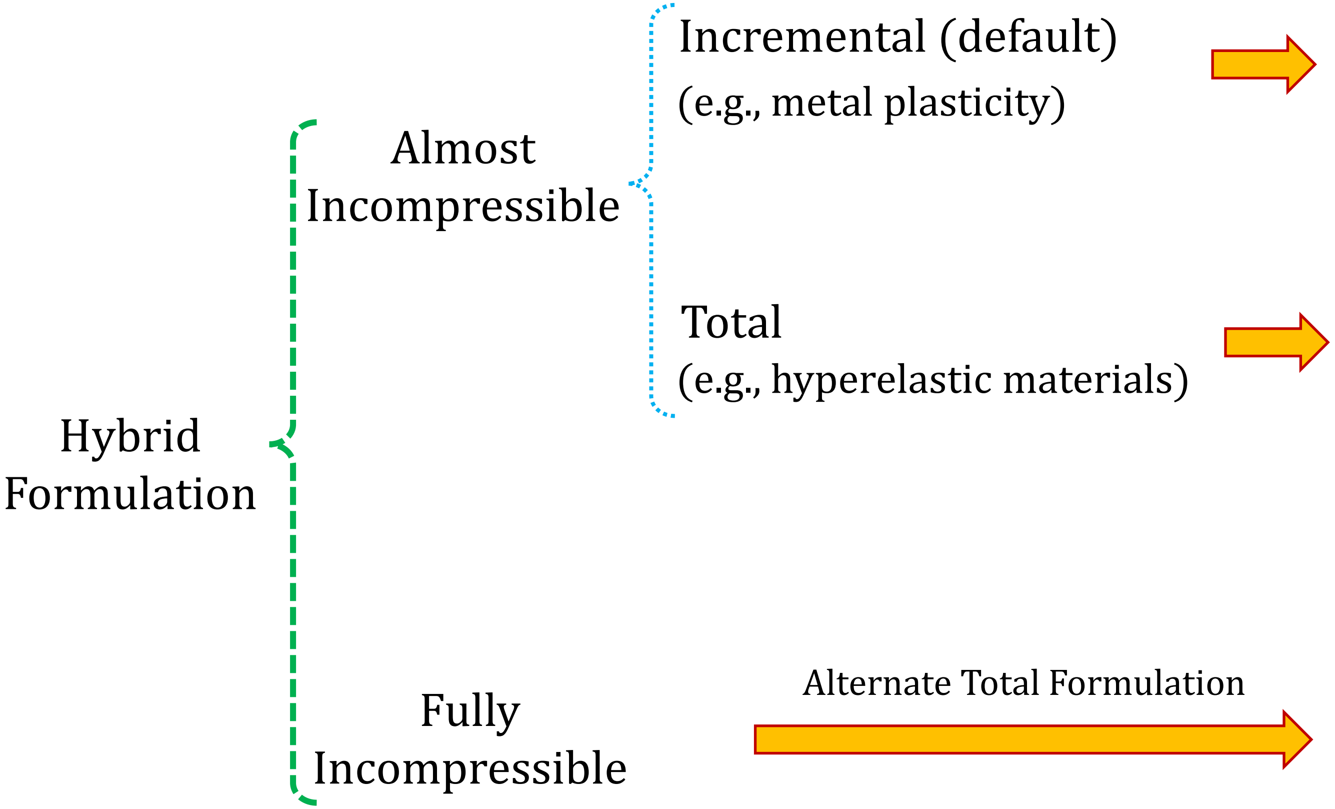
Infinite Stiffness Matrix

For a nearly incompressible material the stiffness matrix become ill conditioned, so that small rounding errors during the computation result in large errors in the solution

Hydrostatic Stress distribution as an additional unknown variable, which must be computed at the same time as the displacement field

Almost Incompressible or Fully Incompressible Elastic Materials

Hybrid Elements



Hybrid Formulation

Almost Incompressible

Incremental (default)

(e.g., metal plasticity)

Total

(e.g., hyperelastic materials)

Fully Incompressible

Alternate Total Formulation

Incremental Lagrange Multiplier–based Formulation

Abaqus/Standard, by default, replaces the **pressure stress** calculated from your definition of STRESS with that derived from the Lagrange multiplier and modifies the Jacobian appropriately

Total Lagrange Multiplier–based Formulation

Assumes that the response of the material can be written as the sum of its **deviatoric** and **volumetric** parts and that these parts are decoupled from each other

Only the deviatoric stress and Jacobian need to be defined for a fully incompressible material response through user subroutine UMAT

Total Hybrid Formulation

The **Total Hybrid Formulation** assumes that the response of the material can be written as the sum of its deviatoric and volumetric parts and that these parts are decoupled from each other

The volumetric part of the strain energy density potential

$$U(\bar{I}_1, \bar{I}_2, \hat{J}) = C_{10}(\bar{I}_1 - 3) + C_{01}(\bar{I}_2 - 3) + \frac{1}{D_1}(\hat{J} - 1)^2$$

Alternate Variable

$$p \overset{\text{def}}{=} -\frac{1}{3}\mathbf{I} : \boldsymbol{\sigma},$$

Stress

$$\mathbf{S} \overset{\text{def}}{=} \boldsymbol{\sigma} + p\,\mathbf{I},$$

Deviatoric Part Of The Stress Tensor

$$\left\{ \begin{array}{l} \mathbf{S} = \frac{2}{J}\,\mathrm{DEV}\left[\left(\frac{\partial U}{\partial \bar{I}_1} + \bar{I}_1 \frac{\partial U}{\partial \bar{I}_2}\right)\bar{\mathbf{B}} - \frac{\partial U}{\partial \bar{I}_2}\bar{\mathbf{B}}\,.\,\bar{\mathbf{B}}\right] \\ \hat{p} = -\frac{\partial U_{vol}}{\partial \hat{J}} \end{array} \right.$$

Hydrostatic/Volumetric Part Of The Stress Tensor

Read only: → STRESS (NTENS+1): $\hat{J}$

Write only: →

STRESS (NTENS+2): $\hat{K} = -J\frac{\partial \hat{p}}{\partial \hat{J}} = J\frac{\partial^2 U_{vol}}{\partial \hat{J}^2} \longrightarrow \hat{K} = J\frac{2}{D_1}$

STRESS (NTENS+3): $\frac{\partial \hat{K}}{\partial \hat{J}} = J\frac{\partial^3 U_{vol}}{\partial \hat{J}^3} \longrightarrow \frac{\partial \hat{K}}{\partial \hat{J}} = 0$

Total Hybrid Formulation

The **Total Hybrid Formulation** assumes that the response of the material can be written as the sum of its deviatoric and volumetric parts and that these parts are decoupled from each other

The volumetric part of the strain energy density potential

Deviatoric Part Of The Stress Tensor

$$U(\bar{I}_1,\bar{I}_2,\hat{J}) = C_{10}(\bar{I}_1-3)+C_{01}(\bar{I}_2-3)+\frac{1}{D_1}(\hat{J}-1)^2 \xrightarrow{\text{Stress}} \begin{cases} \mathbf{S}=\dfrac{2}{J}\,\mathrm{DEV}\left[\left(\dfrac{\partial U}{\partial \bar{I}_1}+\bar{I}_1\dfrac{\partial U}{\partial \bar{I}_2}\right)\bar{\mathbf{B}}-\dfrac{\partial U}{\partial \bar{I}_2}\bar{\mathbf{B}}\,.\,\bar{\mathbf{B}}\right] \\ \hat{p}=-\dfrac{\partial U_{vol}}{\partial \hat{J}} \end{cases}$$

Alternate Variable

Hydrostatic/Volumetric Part Of The Stress Tensor

$$\sigma_{ij}=\frac{2}{J}C_{10}\left(\bar{B}_{ij}-\frac{1}{3}\delta_{ij}\bar{B}_{kk}\right)+\frac{2}{J}C_{01}\left(\bar{B}_{kk}\bar{B}_{ij}-\frac{1}{3}\delta_{ij}\,(\bar{B}_{kk})^2-\bar{B}_{ik}\bar{B}_{kj}+\frac{1}{3}\delta_{ij}\bar{B}_{kn}\bar{B}_{nk}\right)+\frac{2}{D_1}(\hat{J}-1)\delta_{ij}$$

$$C_{ijkl}=\frac{2}{J}C_{10}\left[\frac{1}{\underbrace{2}\,}\underbrace{(\delta_{ik}\bar{B}_{jl}+\bar{B}_{ik}\delta_{jl}+\delta_{il}\bar{B}_{jk}+\bar{B}_{il}\delta_{jk})}_{\delta_{ik}\bar{B}_{jl}+\bar{B}_{il}\delta_{jk}}-\frac{2}{3}\delta_{ij}\bar{B}_{kl}-\frac{2}{3}\bar{B}_{ij}\delta_{kl}+\frac{2}{9}\delta_{ij}\delta_{kl}\bar{B}_{mm}\right]+\underbrace{J\frac{2}{D_1}}_{\hat{K}}\delta_{ij}\delta_{kl}$$

Objectivity and Material Symmetry

The **principle of objectivity** or **material-frame indifference** states that material properties are independent of superimposed rigid-body motions.

For Hyperelastic materials, the principle of objectivity implies that W only depends on $\mathbf{F}$ through $\mathbf{C}$, so that we can write $W(\mathbf{F}) = -W(\mathbf{C})$.

$$\underbrace{W(\mathbf{X}, t) = W\ (F(\mathbf{X}, t), \mathbf{X})}_{\text{Hyperelastic Materials}} = -W\ (\mathbf{C}(\mathbf{X}, t), \mathbf{X})$$

A material is said to be symmetric with respect to a linear transformation if the reference configuration is mapped by this transformation to another configuration which is mechanically indistinguishable from it

Hyperelastic Materials

$$\mathbf{T} = w_0\mathbf{1} + w_1\mathbf{B} + w_2\mathbf{B}^2$$

$$\mathbf{T} = \left(2J\,\frac{\partial W}{\partial \mathbf{I}_3} - p\right)\mathbf{I} + \left(\frac{2}{J}\frac{\partial W}{\partial \mathbf{I}_1} + \frac{2}{J}\frac{\partial W}{\partial \mathbf{I}_2}\mathbf{I}_1\right)\mathbf{B} + \left(-\frac{2}{J}\frac{\partial W}{\partial \mathbf{I}_2}\right)\boldsymbol{B}^{\mathbf{2}}$$

$p = 0$ for compressible materials and $J = I_3 = 1$ for incompressible materials.

$$w_0 = 2J\frac{\partial W}{\partial I_3} - p,$$

$$w_1 = 2J^{-1}\frac{\partial W}{\partial I_1} + 2J^{-1}\frac{\partial W}{\partial I_2}I_1$$

$$w_2 = -2J^{-1}\frac{\partial W}{\partial I_2}.$$

Alternative Representation

Cayley–Hamilton theorem

$$\mathbf{T} = \beta_0\mathbf{1} + \beta_1\mathbf{B} + \beta_{-1}\mathbf{B}^{-1}$$

$$\mathbf{T} = \left(2J\,\frac{\partial W}{\partial \mathbf{I}_3} - \frac{2\mathbf{I}_2}{J}\frac{\partial W}{\partial \mathbf{I}_2} - p\right)\mathbf{I} + \left(\frac{2}{J}\frac{\partial W}{\partial \mathbf{I}_1}\right)\mathbf{B} + \left(-2\frac{\partial W}{\partial \mathbf{I}_2}\right)\boldsymbol{B}^{\mathbf{-1}}$$

$p = 0$ for compressible materials and $J = I_3 = 1$ for incompressible materials.

$$\beta_0 = 2J\frac{\partial W}{\partial I_3} + 2J^{-1}I_2\frac{\partial W}{\partial I_2} - p$$

$$\beta_1 = 2J^{-1}\frac{\partial W}{\partial I_1},$$

$$\beta_{-1} = -2J\frac{\partial W}{\partial I_2}.$$

Choice of Strain-Energy Functions

Incompressible
Neo-Hookean Materials

$$W_{\mathrm{nh}} = \frac{C_1}{2}(I_1 - 3)$$

$$E = 3\mu = 3C_1$$

Incompressible
Mooney–Rivlin
Materials

$$W_{\mathrm{mr}} = \frac{C_1}{2}(I_1 - 3) + \frac{C_2}{2}(I_2 - 3)$$

$$C_1 + C_2 = \mu$$

$$C_1 = \mu(\frac{1}{2} + \alpha), \quad C_2 = \mu(\frac{1}{2} - \alpha)$$

$$\alpha \in [-1/2, 1/2]$$

Incompressible
Ogden Materials

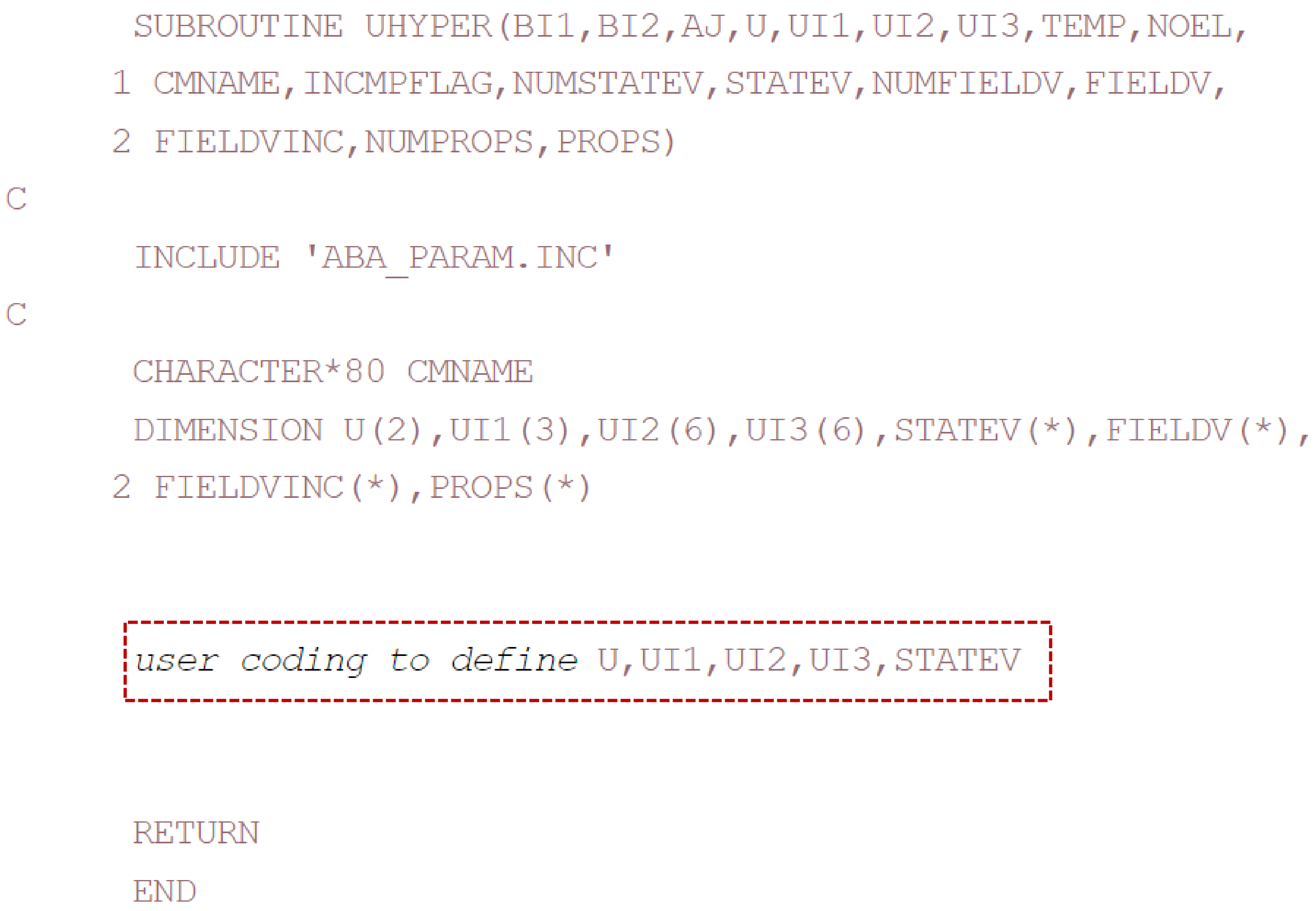
$$W_{\mathrm{og}N} = \sum_{i=1}^{N} \frac{\mu_i}{\alpha_i} \left(\lambda_1^{\alpha_i} + \lambda_2^{\alpha_i} + \lambda_3^{\alpha_i} - 3\right)$$

Incompressible
Fung–Demiray Materials

$$W_{\mathrm{fu}} = \frac{\mu}{2\beta}[\exp(\beta(I_1 - 3)) - 1]$$

UHYPER

Abaqus User Subroutines To Define a Hyperelastic Material



```
      SUBROUTINE UHYPER(BI1,BI2,AJ,U,UI1,UI2,UI3,TEMP,NOEL,
     1 CMNAME,INCMPFLAG,NUMSTATEV,STATEV,NUMFIELDV,FIELDV,
     2 FIELDVINC,NUMPROPS,PROPS)
C
      INCLUDE 'ABA_PARAM.INC'
C
      CHARACTER*80 CMNAME
      DIMENSION U(2),UI1(3),UI2(6),UI3(6),STATEV(*),FIELDV(*),
     2 FIELDVINC(*),PROPS(*)


      user coding to define U,UI1,UI2,UI3,STATEV


      RETURN
      END
```

Hyperelastic Material

Volume-preserving, Or Isochoric Part of **F**

Deformation Gradient

$$\mathbf{F} = \nabla_0 \mathbf{x} = \frac{\partial \mathbf{x}\,(\mathbf{X}, t)}{\partial \mathbf{X}}$$

⇒

Distortion Gradient

$$\bar{\mathbf{F}} = J^{-\frac{1}{3}}\,\mathbf{F}$$

Jacobian Determinant

⇒

$$\left\{ \begin{array}{l} \bar{\mathbf{C}} = \bar{\mathbf{F}}^{T}.\,\bar{\mathbf{F}} \\ \\ \bar{\mathbf{B}} = \bar{\mathbf{F}}\,.\,\bar{\mathbf{F}}^{T} \end{array} \right.$$

Deviatoric Right Cauchy-green Deformation Tensor

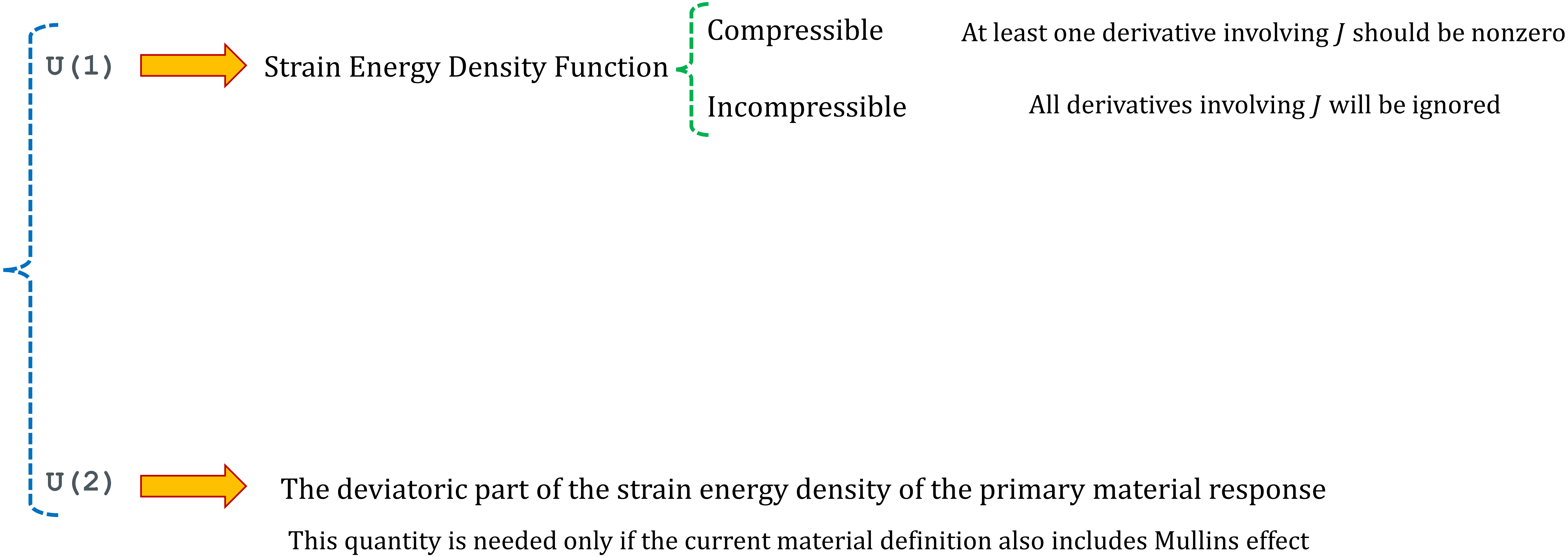
Deviatoric Left Cauchy-green Deformation Tensor

$$U = f(I_1, I_2, I_3)$$

⇒

$$U = \underset{f(\bar{I}_1, \bar{I}_2)}{\bar{U}_{deviatoric}} + \underset{f\left(J = \sqrt{I_3}\,\right)}{U_{hydrostatic}}$$

Variables to Be Defined



U(1)

Strain Energy Density Function

Compressible

At least one derivative involving J should be nonzero

Incompressible

All derivatives involving J will be ignored

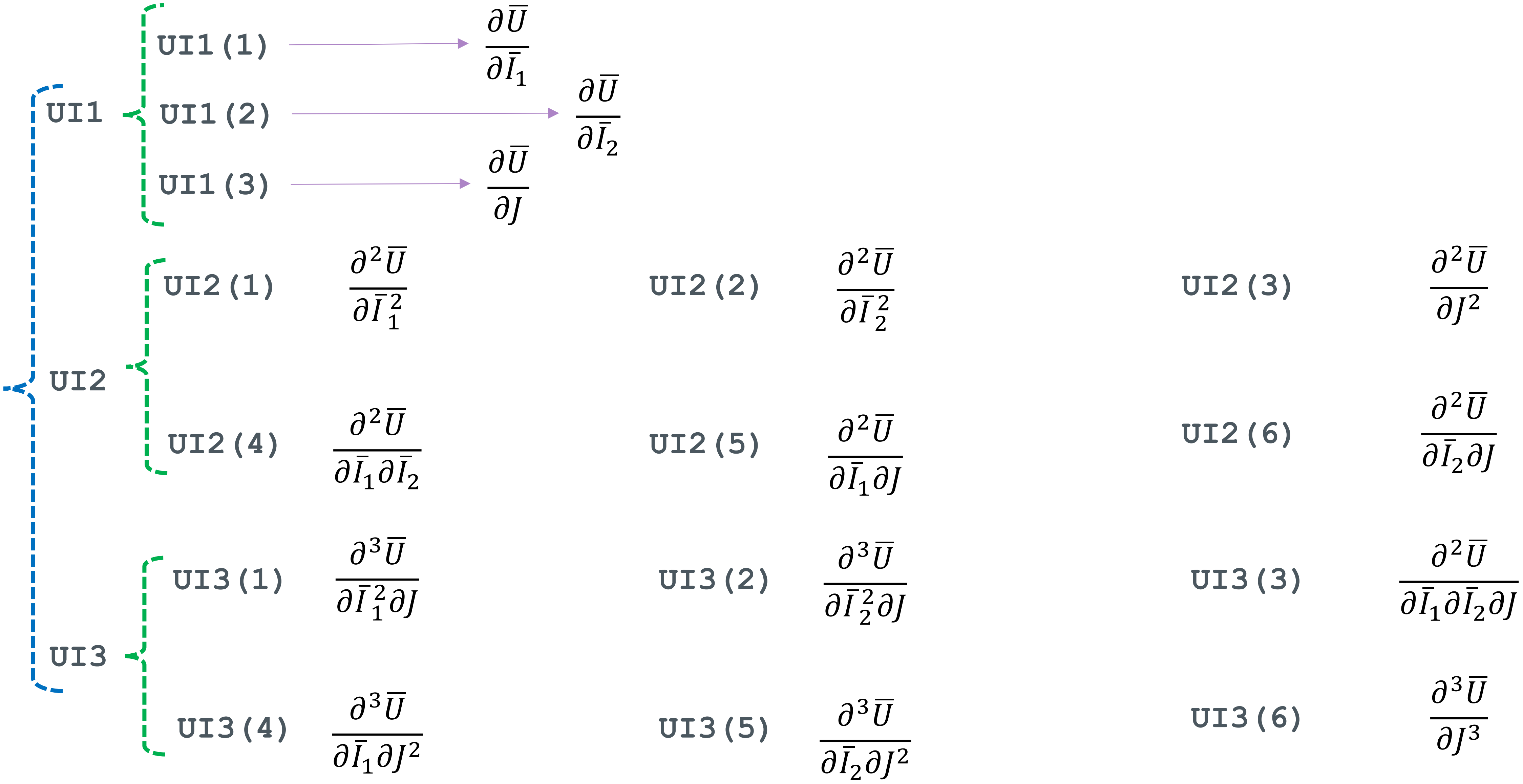
U(2)

The deviatoric part of the strain energy density of the primary material response

This quantity is needed only if the current material definition also includes Mullins effect

Mullins Effect

Variables Passed in for Information



UI1

UI1(1) → $\frac{\partial \bar{U}}{\partial \bar{I}_1}$

UI1(2) → $\frac{\partial \bar{U}}{\partial \bar{I}_2}$

UI1(3) → $\frac{\partial \bar{U}}{\partial J}$

UI2

UI2(1) $\frac{\partial^2 \bar{U}}{\partial \bar{I}_1^2}$

UI2(2) $\frac{\partial^2 \bar{U}}{\partial \bar{I}_2^2}$

UI2(3) $\frac{\partial^2 \bar{U}}{\partial J^2}$

UI2(4) $\frac{\partial^2 \bar{U}}{\partial \bar{I}_1 \partial \bar{I}_2}$

UI2(5) $\frac{\partial^2 \bar{U}}{\partial \bar{I}_1 \partial J}$

UI2(6) $\frac{\partial^2 \bar{U}}{\partial \bar{I}_2 \partial J}$

UI3

UI3(1) $\frac{\partial^3 \bar{U}}{\partial \bar{I}_1^2 \partial J}$

UI3(2) $\frac{\partial^3 \bar{U}}{\partial \bar{I}_2^2 \partial J}$

UI3(3) $\frac{\partial^2 \bar{U}}{\partial \bar{I}_1 \partial \bar{I}_2 \partial J}$

UI3(4) $\frac{\partial^3 \bar{U}}{\partial \bar{I}_1 \partial J^2}$

UI3(5) $\frac{\partial^3 \bar{U}}{\partial \bar{I}_2 \partial J^2}$

UI3(6) $\frac{\partial^3 \bar{U}}{\partial J^3}$

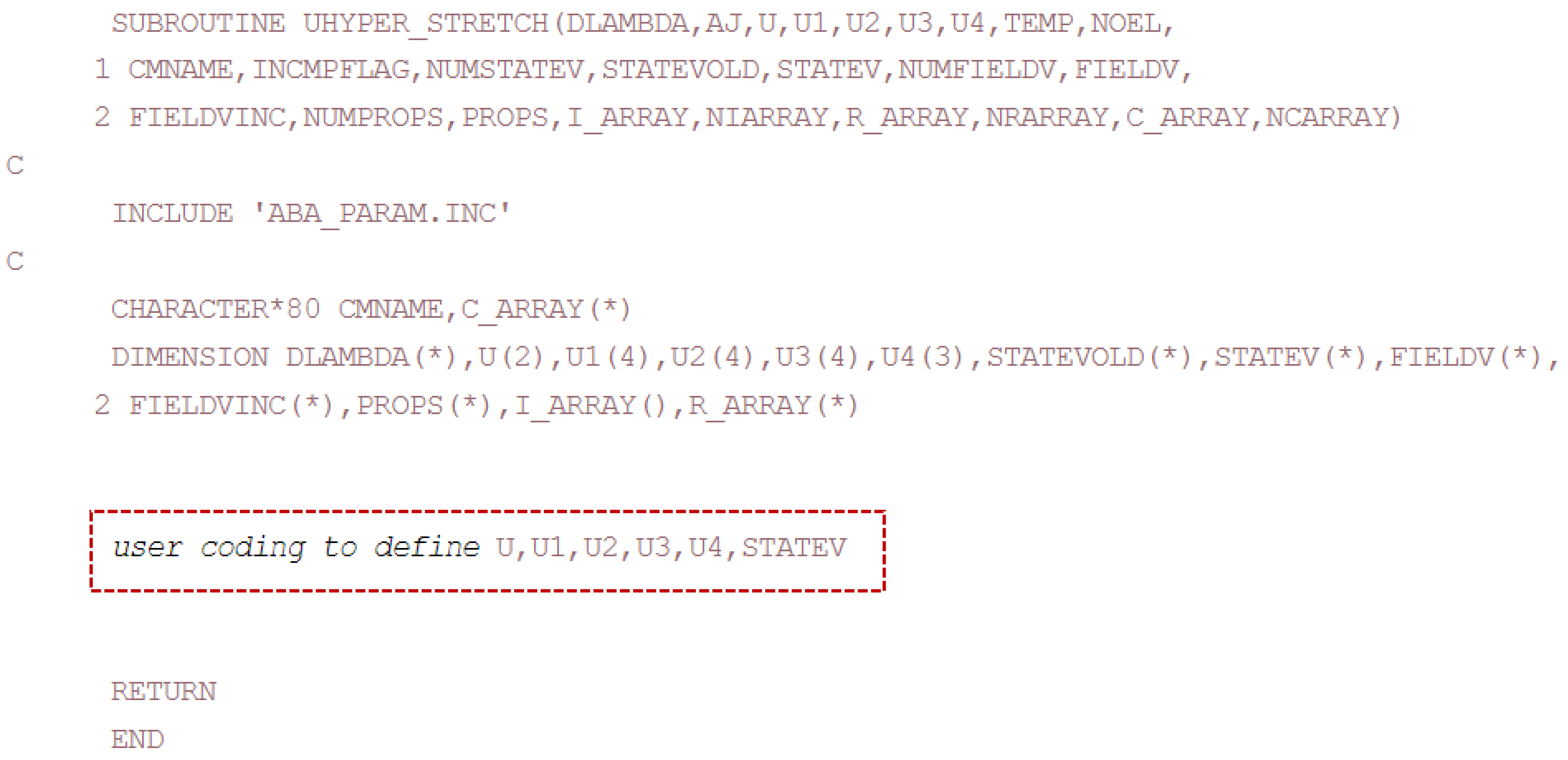
Variables Passed in for Information

STATEV

Array containing the user-defined solution-dependent state variables at this point. These are supplied as values at the start of the increment or as values updated by other user subroutines and must be returned as values at the end of the increment.

UHYPER_STRETCH

Abaqus User Subroutines To Define a Hyperelastic Material in Term of Principal Stretches

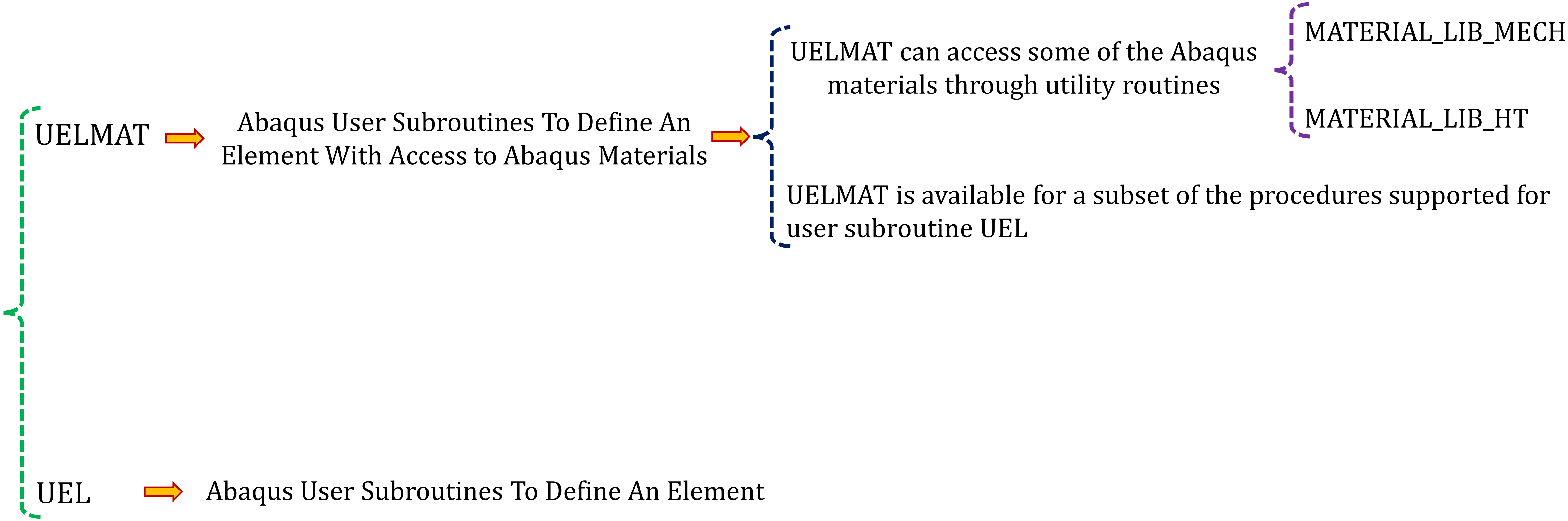


```
       SUBROUTINE UHYPER_STRETCH(DLAMBDA,AJ,U,U1,U2,U3,U4,TEMP,NOEL,
      1 CMNAME,INCMPFLAG,NUMSTATEV,STATEVOLD,STATEV,NUMFIELDV,FIELDV,
      2 FIELDVINC,NUMPROPS,PROPS,I_ARRAY,NIARRAY,R_ARRAY,NRARRAY,C_ARRAY,NCARRAY)
C
       INCLUDE 'ABA_PARAM.INC'
C
       CHARACTER*80 CMNAME,C_ARRAY(*)
       DIMENSION DLAMBDA(*),U(2),U1(4),U2(4),U3(4),U4(3),STATEVOLD(*),STATEV(*),FIELDV(*),
      2 FIELDVINC(*),PROPS(*),I_ARRAY(),R_ARRAY(*)


       user coding to define U,U1,U2,U3,U4,STATEV


       RETURN
       END
```

Variables to Be Defined

Variables Passed in for Information

User-defined Element



UELMAT → Abaqus User Subroutines To Define An Element With Access to Abaqus Materials →

- UELMAT can access some of the Abaqus materials through utility routines
  - MATERIAL_LIB_MECH
  - MATERIAL_LIB_HT
- UELMAT is available for a subset of the procedures supported for user subroutine UEL

UEL → Abaqus User Subroutines To Define An Element

UELMAT

Abaqus User Subroutines To Define An (Nonlinear) Element With Access to Abaqus Materials

```
...

*USER ELEMENT, TYPE=U1, NODES=#, COORDINATES=#, PROPERTIES=#, I PROPERTIES=#,
VARIABLES=#, UNSYMM, INTEGRATION=#, TENSOR=.. .
```

Data line(s)

Number of element integration points

Specifies the element type

- THREED (3D stress/displacement or heat transfer)
- TWOD (2D heat transfer)
- PSTRAIN (plane strain)
- PSTRESS (plane stress)

```
*ELEMENT, TYPE=U1, ELSET=SOLID
```

Data line(s)

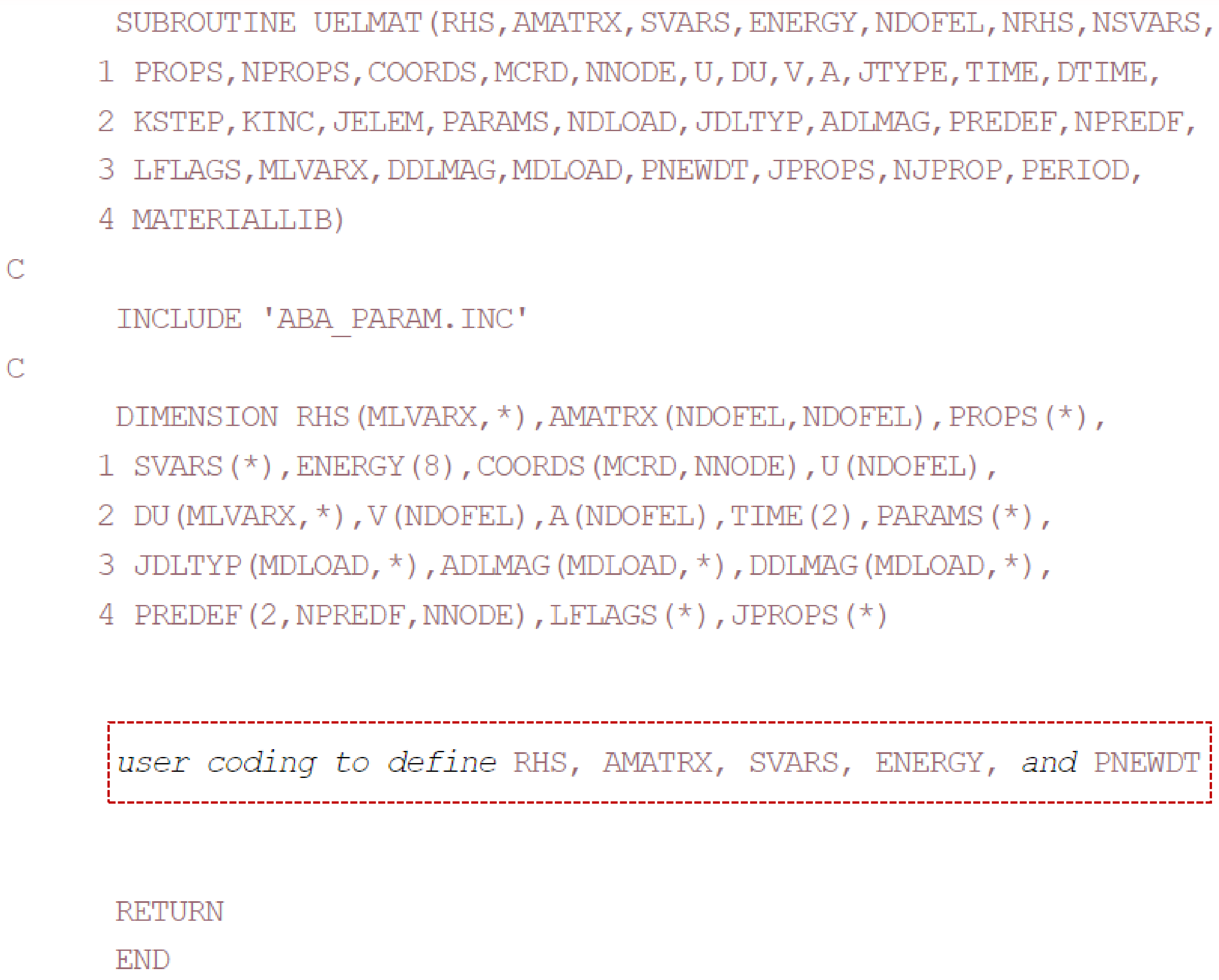
```
*UEL PROPERTY, ELSET=SOLID, MATERIAL=MAT
```

Data line(s)

```
*MATERIAL, NAME=MAT
```

UELMAT

Abaqus User Subroutines To Define An (Nonlinear) Element With Access to Abaqus Materials



```
      SUBROUTINE UELMAT(RHS,AMATRX,SVARS,ENERGY,NDOFEL,NRHS,NSVARS,
     1 PROPS,NPROPS,COORDS,MCRD,NNODE,U,DU,V,A,JTYPE,TIME,DTIME,
     2 KSTEP,KINC,JELEM,PARAMS,NDLOAD,JDLTYP,ADLMAG,PREDEF,NPREDF,
     3 LFLAGS,MLVARX,DDLMAG,MDLOAD,PNEWDT,JPROPS,NJPROP,PERIOD,
     4 MATERIALLIB)
C
      INCLUDE 'ABA_PARAM.INC'
C
      DIMENSION RHS(MLVARX,*),AMATRX(NDOFEL,NDOFEL),PROPS(*),
     1 SVARS(*),ENERGY(8),COORDS(MCRD,NNODE),U(NDOFEL),
     2 DU(MLVARX,*),V(NDOFEL),A(NDOFEL),TIME(2),PARAMS(*),
     3 JDLTYP(MDLOAD,*),ADLMAG(MDLOAD,*),DDLMAG(MDLOAD,*),
     4 PREDEF(2,NPREDF,NNODE),LFLAGS(*),JPROPS(*)


      user coding to define RHS, AMATRX, SVARS, ENERGY, and PNEWDT


      RETURN
      END
```

Variables Passed in for Information

DTIME → Time increment

PERIOD → Time period of the current step

NDOFEL → Number of degrees of freedom in the element

MLVARX → Dimensioning parameter used when **several displacement** or **right-hand-side** vectors are used

```
RHS(MLVARX,*), DU(MLVARX,*)
```

NRHS → Number of load vectors

- NRHS=1 in most nonlinear problems
- NRHS=2 for the modified Riks static procedure
- Greater than 1 in some linear analysis procedures and during substructure generation

⇩

For example, in the recovery path for the **direct steady-state** procedure, it is 2 to accommodate the **real** and **imaginary** parts of the vectors

Variables Passed in for Information

NSVARS ---> User-defined **number of solution-dependent state variables** associated with the element

NPROPS ---> User-defined **number of real property** values associated with the element

NJPROP ---> User-defined **number of integer property** values associated with the element

MCRD <= 3 ---> The maximum of
- Maximum number of coordinates required at any node point
- Value of the largest active degree of freedom

NNODE ---> User-defined **number of nodes on the element**

Variables Passed in for Information

JTYPE → Integer defining the element type(n)

Abaqus/Standard	Un	$(n \leq 10000)$
Abaqus/Explicit	VUn	$(n \leq 9000)$

KSTEP → Current step number

KINC → Current increment number

JELEM → User-assigned element number

NDLOAD → Identification number of the distributed load or flux currently **active** on this element

MDLOAD → Total number of distributed loads and/or fluxes defined on this element

NPREDF → Number of predefined field variables, including temperature
For user elements Abaqus/Standard uses one value for each field variable per node

Variables Passed in for Information

MATERIALLIB → A variable that must be passed to the utility routines performing material point computations

Accessing Abaqus Materials

MATERIAL_LIB_MECH

```
 DIMENSION STRESS(*),DDSDDE(NTENS,*),STRAN(*),DSTRAN(*),
*        DEFGRAD(3,3),PREDEF(NPREDF),DPREDEF(NPREDF),COORDS(3)
     ...

 CALL MATERIAL_LIB_MECH(MATERIALLIB,STRESS,DDSDDE,STRAN,DSTRAN,
*       NPT,DVDV0,DVMAT,DFGRD,PREDEF,DPREDEF,NPREDF,CELENT,COORDS)
     ...
```

MATERIAL_LIB_HT

```
 DIMENSION PREDEF(NPREDEF),DPREDEF(NPREDEF),DTEMDX(*),
*       RHODUDG(*),FLUX(*),DFDT(*),DFDG(NDIM,*),DRPLDT(*),
*       COORDS(3)
     ...

 CALL MATERIAL_LIB_HT(MATERIALLIB,RHOUDOT,RHODUDT,RHODUDG,
*       FLUX,DFDT,DFDG,RPL,DRPLDT,NPT,DVMAT,PREDEF,
*       DPREDEF,NPREDF,TEMP,DTEMP,DTEMDX,CELENT,COORDS)
     ...
```

Variables Passed in for Information

PROPS(*) → A **floating point** array containing the NPROPS real property values defined for use with this element. NPROPS is the user-specified number of real property values

JPROPS(*) → An **integer** array containing the NJPROP **integer** property values defined for use with this element. NJPROP is the user-specified number of integer property values

COORDS(MCRD, NNODE) → An array containing the **original coordinates** of the nodes of the element COORDS(K1,K2) is the $K1^{th}$ coordinate of the $K2^{th}$ node of the element

JDLTYP(*) → An array containing the **integers** used to define **distributed load types** for the element

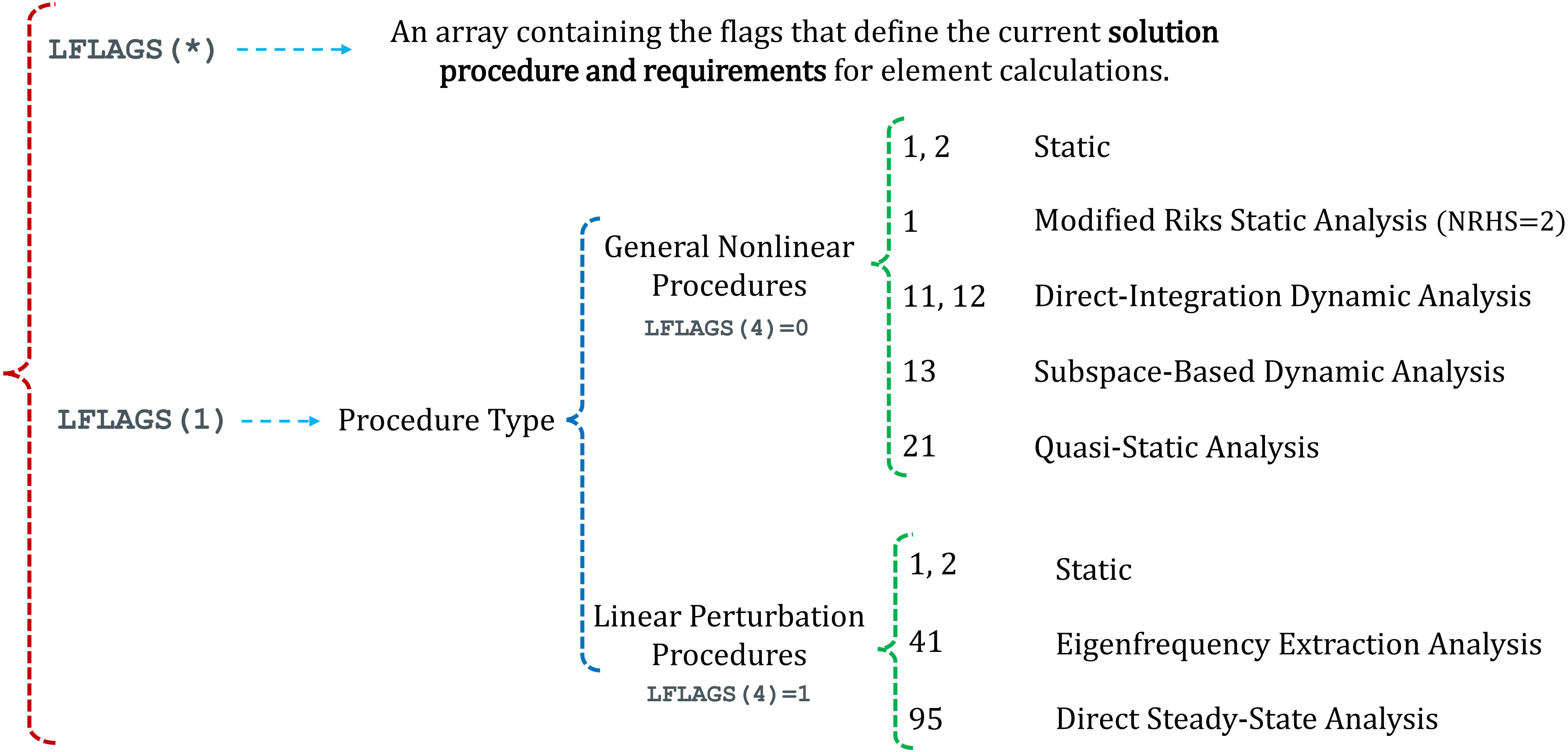
Loads of type Un are identified by the integer value n in JDLTYP

Loads of type UnNU are identified by the negative integer value −n in JDLTYP

JDLTYP(K1,K2) is the identifier of the $K1^{th}$ distributed load in the $K2^{th}$ load case

For general nonlinear steps: K2 =1

Variables Passed in for Information



LFLAGS(*) → An array containing the flags that define the current **solution procedure and requirements** for element calculations.

LFLAGS(1) → Procedure Type

- General Nonlinear Procedures LFLAGS(4)=0

1, 2	Static
1	Modified Riks Static Analysis (NRHS=2)
11, 12	Direct-Integration Dynamic Analysis
13	Subspace-Based Dynamic Analysis
21	Quasi-Static Analysis

- Linear Perturbation Procedures LFLAGS(4)=1

1, 2	Static
41	Eigenfrequency Extraction Analysis
95	Direct Steady-State Analysis

Variables Passed in for Information

LFLAGS (1)	Procedure	Comments
1, 2	Static	Automatic/fixed time incrementation
11,12	Dynamic	Automatic/fixed time incrementation
21,22	Visco	Quasi-static; explicit/implicit time integration
31	Heat Transfer	Steady-state
32, 33	Heat Transfer	Transient; fixed/automatic time incrementation
41	Frequency extraction	
61	Geostatic	
62, 63	Soils	Steady-state; fixed/automatic time incrementation
64, 65	Soils	Transient; fixed/automatic time incrementation
71	Coupled thermal-stress	Steady-state
72,73	Coupled thermal-stress	Transient; fixed/automatic time incrementation
75	Coupled thermal-electrical	Steady-state
76,77	Coupled thermal-electrical	Transient; fixed/automatic time incrementation

Variables Passed in for Information

`LFLAGS(*)` → An array containing the flags that define the current **solution procedure and requirements** for element calculations.

`LFLAGS(2)=`

- 0 Small-displacement analysis
- 1 Large-displacement analysis (nonlinear geometric effects included in the step)

Variables Passed in for Information

`LFLAGS(3)=`

1 Normal implicit time incrementation procedure. User subroutine UEL must define the residual vector in RHS and the Jacobian matrix in AMATRX.

2 Define the current stiffness matrix (AMATRX = $K^{NM} = -\frac{\partial F^N}{\partial u^M}$ or $-\frac{\partial G^N}{\partial u^M}$) only

3 Define the current damping matrix (AMATRX = $C^{NM} = -\frac{\partial F^N}{\partial \dot{u}^{\mathrm{M}}}$ or $-\frac{\partial G^N}{\partial \dot{u}^{\mathrm{M}}}$) only

4 Define the current mass matrix (AMATRX = $M^{NM} = -\frac{\partial F^N}{\partial \ddot{u}^M}$) only.

Abaqus/Standard always requests an initial mass matrix at the start of the analysis.

5 Define the **current residual or load vector** (RHS =F^N) only

6 Define the current **mass matrix** and the **residual vector** for the initial acceleration calculation (or the calculation of accelerations after impact)

100 Define perturbation quantities for output.
Not available for direct steady-state dynamic and mode-based procedures

Variables Passed in for Information

`LFLAGS(4)=`

0 The step is a general step

1 The step is a linear perturbation step

`LFLAGS(5)=`

0 The current approximations to u^M, etc. were based on **Newton corrections**

1 The current approximations were found by **extrapolation** from the previous increment

`LFLAGS(7)=`

1 When the damping matrix flag is set, the **viscous damping** matrix is defined

2 When the damping matrix flag is set, the **structural damping** matrix is defined

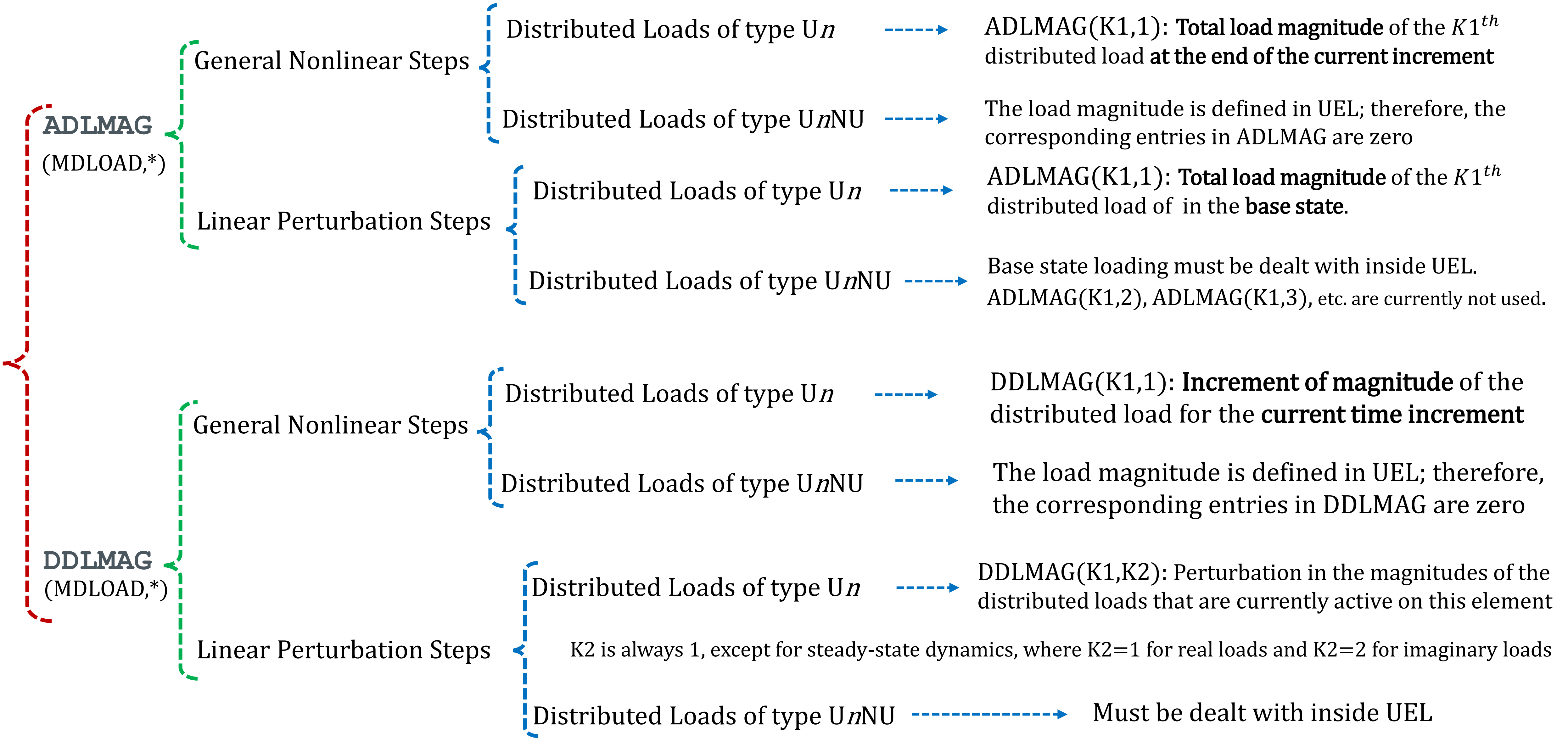
Variables Passed in for Information

```
U, V, A (NDOFEL)
DU(MLVARX,*)
```

Arrays containing the current estimates of the **basic solution variables** (displacements, rotations, temperatures, depending on the degree of freedom) at the nodes of the element at the **end of the current incremen**t. Values are provided as follows:

$U\ (K1\ )$	Total values of the variables. If this is a linear perturbation step, it is the value in the **base state**.
$DU\ (K1, KRHS)$	Incremental values of the variables for the current increment for right-hand-side KRHS. For eigenvalue extraction step, this is the eigenvector magnitude for eigenvector KRHS. For steady-state dynamics, KRHS = 1 denotes real components of perturbation displacement and KRHS = 2 denotes imaginary components of perturbation displacement.
$V\ (K1\ )$	Time rate of change of the variables (velocities, rates of rotation). Defined for implicit dynamics only (LFLAGS (1) = 11 or 12).
$A\ (K1\ )$	Accelerations of the variables. Defined for implicit dynamics only (LFLAGS (1) = 11 or 12).

Variables Passed in for Information

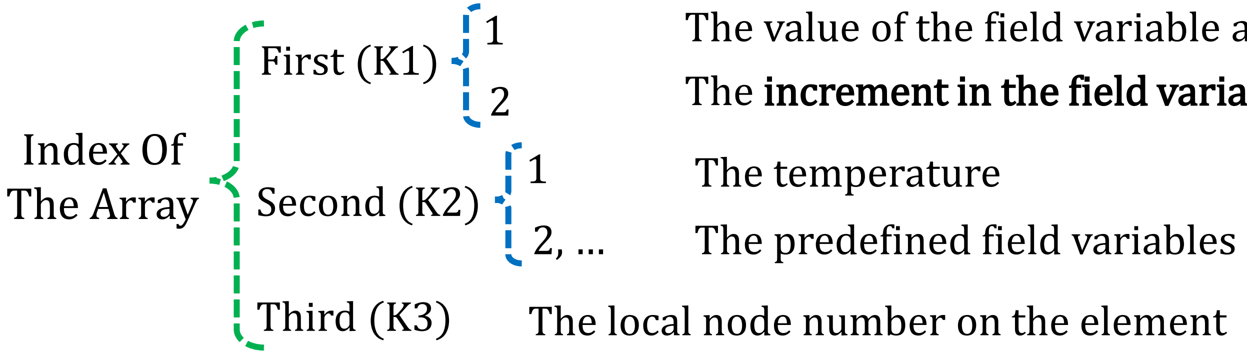


- **ADLMAG** (MDLOAD,*)
  - General Nonlinear Steps
    - Distributed Loads of type U*n* → ADLMAG(K1,1): **Total load magnitude** of the $K1^{th}$ distributed load **at the end of the current increment**
    - Distributed Loads of type U*n*NU → The load magnitude is defined in UEL; therefore, the corresponding entries in ADLMAG are zero
  - Linear Perturbation Steps
    - Distributed Loads of type U*n* → ADLMAG(K1,1): **Total load magnitude** of the $K1^{th}$ distributed load of in the **base state**.
    - Distributed Loads of type U*n*NU → Base state loading must be dealt with inside UEL. ADLMAG(K1,2), ADLMAG(K1,3), etc. are currently not used.
- **DDLMAG** (MDLOAD,*)
  - General Nonlinear Steps
    - Distributed Loads of type U*n* → DDLMAG(K1,1): **Increment of magnitude** of the distributed load for the **current time increment**
    - Distributed Loads of type U*n*NU → The load magnitude is defined in UEL; therefore, the corresponding entries in DDLMAG are zero
  - Linear Perturbation Steps
    - Distributed Loads of type U*n* → DDLMAG(K1,K2): Perturbation in the magnitudes of the distributed loads that are currently active on this element
    - K2 is always 1, except for steady-state dynamics, where K2=1 for real loads and K2=2 for imaginary loads
    - Distributed Loads of type U*n*NU → Must be dealt with inside UEL

Variables Passed in for Information

`PREDEF(2,NPREDF,NNODE)`

An array containing the values of predefined field variables, such as temperature in an uncoupled stress/displacement analysis, at the nodes of the element



Index Of The Array

First (K1)
- 1 — The value of the field variable at the **end of the increment**
- 2 — The **increment in the field variable**

Second (K2)
- 1 — The temperature
- 2, … — The predefined field variables

Third (K3) — The local node number on the element

In cases where temperature is not defined, the predefined field variables begin with index 1

PREDEF (K1,1,K3)	Temperature.
PREDEF (K1,2, ,K3)	First predefined field variable.
PREDEF (K1,3, K3)	Second predefined field variable.
Etc.	Any other predefined field variable.
PREDEF (K1,K2, K3)	Total or incremental value of the $K2^{th}$ predefined field variable at the $K3^{th}$ node of the element.
PREDEF (1,K2,K3)	Values of the variables at the end of the current increment.
PREDEF (2,K2,K3)	Incremental values corresponding to the current time increment.

Variables Passed in for Information

PARAMS(*)

An array containing the parameters associated with the **solution procedure**. The entries in this array depend on the solution procedure currently being used when UEL is called, as indicated by the entries in the LFLAGS array.

For implicit dynamics (LFLAGS(1) = 11 or 12) PARAMS contains the **integration operator values**, as:

PARAMS

- PARAMS(1) → α
- PARAMS(2) → β
- PARAMS(3) → γ

TIME(1) Current value of step time or frequency

TIME(2) Current value of total time

Variables to Be Defined

These arrays depend on the value of the **LFLAGS** array

`RHS(MLVARX,*)` → An array containing the contributions of this element to the right-hand-side vectors of the overall system of equations

`AMATRX(NDOFEL,NDOFEL)` → An array containing the contribution of this element to the Jacobian (stiffness) or other matrix of the overall system of equations

At time Increment n+1

$$\overset{\text{Residual}}{\mathbf{R}}\left(\mathbf{d}^{\mathbf{n+1}}, t^{n+1}\right) = \mathbf{F}_{ext}\left(\mathbf{d}^{\mathbf{n+1}}, t^{n+1}\right) - \mathbf{F}_{int}\left(\mathbf{d}^{\mathbf{n+1}}, t^{n+1}\right) = 0$$

Linearized Model Of The Nonlinear Equations

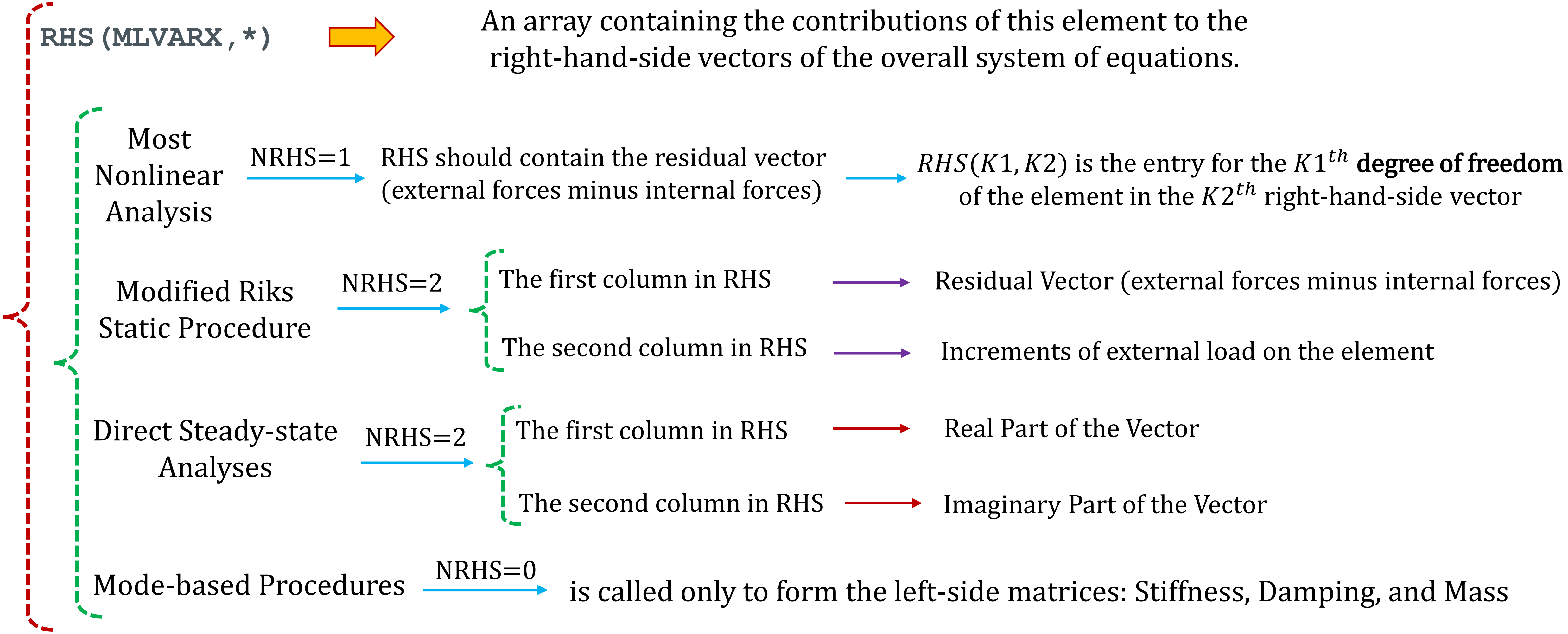
At time Increment n+1
At Iteration m

$$\mathbf{R}(\mathbf{d}_{m+1}, t^{n+1}) = \mathbf{R}(\mathbf{d}_m, t^{n+1}) + \underbrace{\frac{\partial \mathbf{R}(\mathbf{d}_m, t^{n+1})}{\partial \mathbf{d}}}_{\text{Jacobian Matrix}} \underbrace{(\mathbf{d}_{\mathrm{m+1}} - \mathbf{d}_{\mathrm{m}})}_{\Delta\mathbf{d}} = 0$$

$$\underbrace{\frac{\partial \mathbf{R}(\mathbf{d}_m, t^{n+1})}{\partial \mathbf{d}}}_{\substack{\text{NDOFEL× NDOFEL} \\ \downarrow \\ \textbf{AMATRX}}} \Delta\mathbf{d} = \underbrace{\mathbf{R}(\mathbf{d}_m, t^{n+1})}_{\substack{\text{DU(MLVARX,*)} \\ \downarrow \\ \textbf{RHS}}}$$

Variables to Be Defined

These arrays depend on the value of the **LFLAGS** array



`RHS(MLVARX,*)` → An array containing the contributions of this element to the right-hand-side vectors of the overall system of equations.

Most Nonlinear Analysis → NRHS=1 → RHS should contain the residual vector (external forces minus internal forces) → $RHS(K1, K2)$ is the entry for the $K1^{th}$ **degree of freedom** of the element in the $K2^{th}$ right-hand-side vector

Modified Riks Static Procedure → NRHS=2

- The first column in RHS → Residual Vector (external forces minus internal forces)
- The second column in RHS → Increments of external load on the element

Direct Steady-state Analyses → NRHS=2

- The first column in RHS → Real Part of the Vector
- The second column in RHS → Imaginary Part of the Vector

Mode-based Procedures → NRHS=0 → is called only to form the left-side matrices: Stiffness, Damping, and Mass

Variables to Be Defined

These arrays depend on the value of the **LFLAGS** array

AMATRX(NDOFEL,NDOFEL) → An array containing the contribution of this element to the Jacobian (stiffness) or other matrix of the overall system of equations

The particular matrix required at any time depends on the entries in the LFLAGS array

All nonzero entries in AMATRX should be defined, even if the matrix is symmetric

- The matrix is unsymmetric → AMATRX
- The matrix is symmetric → AMATRX $= \frac{1}{2}([A] + [A]^T)$

Variables to Be Defined

These arrays depend on the value of the **LFLAGS** array

SVARS(*) → **An array** containing the values of the **solution-dependent state variables** associated with this element

The number of such variables is **NSVARS**

General Nonlinear Steps → This array is passed into UEL containing the values of these variables at the start of the current increment. They should be updated to be the values at the end of the increment, unless the procedure during which UEL is being called does not require such an update.

Linear Perturbation Steps → This array is passed into UEL containing the values of these variables in the **base state**. They should be returned containing perturbation values if you want to output such quantities.

When KINC is equal to zero, the call to UEL is made for zero increment output.
In this case the values returned will be used only for output purposes and are not updated permanently.

Variables to Be Defined

These arrays depend on the value of the **LFLAGS** array

`ENERGY(8)`

General Nonlinear Steps → ENERGY contains the values of the energy quantities associated with the element

⇩

The values in this array when UEL is called are the element energy quantities at the start of the current increment. They should be updated to the values at the **end of the current increment**

Linear Perturbation Steps → ENERGY contains the values of the energy in the **base state**

⇩

They should be returned containing perturbation values if you wish to output such quantities

Mode-based Procedures ⇨ They are not available for updates

Variables to Be Defined

These arrays depend on the value of the **LFLAGS** array

`ENERGY(1)`	Kinetic energy
`ENERGY(2)`	Elastic strain energy
`ENERGY(3)`	Creep dissipation
`ENERGY(4)`	Plastic dissipation
`ENERGY(5)`	Viscous dissipation
`ENERGY(6)`	“Artificial strain energy”
`ENERGY(7)`	Electrostatic energy
`ENERGY(8)`	Incremental work done by loads applied within the user element

When KINC is equal to zero, the call to UEL is made for zero increment output. In this case the energy values returned will be used only for output purposes and are not updated permanently.

Associated with such effects as artificial stiffness introduced to control hourglassing or other singular modes in the element.

Variables That Can Be Updated

PNEWDT ⇒ Ratio of suggested new time increment to the time increment currently being used (DTIME)

- If automatic time incrementation is chosen → This variable allows you to provide input to the automatic time incrementation algorithms in Abaqus/Standard

  ⇓

  It is useful only during **equilibrium iterations** with the normal time incrementation (LFLAGS(3)=1)

  ⇓

  During a **severe discontinuity iteration** (such as contact changes), PNEWDT is ignored unless CONVERT SDI=YES is specified for this step

- If automatic time incrementation is not selected in the analysis procedure → for all calls to user subroutines for this iteration and the increment converges in this iteration
  - $PNEWDT > 1.0$ → Will be ignored
  - $PNEWDT < 1.0$ → Will cause the job to terminate

Variables That Can Be Updated

If Automatic Time Incrementation Is Chosen:

If PNEWDT is redefined to be less than 1.0

Abaqus/Standard **must** abandon the time increment and attempt it again with a smaller time increment. The suggested new time increment provided to the automatic time integration algorithms is PNEWDT × DTIME, where the PNEWDT used is the minimum value for all calls to user subroutines that allow redefinition of PNEWDT for this iteration

If PNEWDT is given a value that is greater than 1.0

(For all calls to user subroutines for this iteration and the increment converges in this iteration)

Abaqus/Standard **may** increase the time increment. The suggested new time increment provided to the automatic time integration algorithms is PNEWDT × DTIME, where the PNEWDT used is the minimum value for all calls to user subroutines for this iteration.

Accessing Abaqus Materials

MATERIAL_LIB_MECH

```
       DIMENSION STRESS(*),DDSDDE(NTENS,*),STRAN(*),DSTRAN(*),
     *         DFGRD(3,3),PREDEF(NPREDF),DPREDEF(NPREDF),COORDS(3)
          ...

       CALL MATERIAL_LIB_MECH(MATERIALLIB,STRESS,DDSDDE,STRAN,DSTRAN,
     *        NPT,DVDV0,DVMAT,DFGRD,PREDEF,DPREDEF,NPREDF,CELENT,COORDS)
          ...
```

MATERIAL_LIB_HT

```
       DIMENSION PREDEF(NPREDEF),DPREDEF(NPREDEF),DTEMDX(*),
     *        RHODUDG(*),FLUX(*),DFDT(*),DFDG(NDIM,*),DRPLDT(*),
     *        COORDS(3)
          ...

       CALL MATERIAL_LIB_HT(MATERIALLIB,RHOUDOT,RHODUDT,RHODUDG,
     *        FLUX,DFDT,DFDG,RPL,DRPLDT,NPT,DVMAT,PREDEF,
     *        DPREDEF,NPREDF,TEMP,DTEMP,DTEMDX,CELENT,COORDS)
          ...
```

MATERIAL_LIB_MECH

Returns the **stress** and the **material Jacobian** at the element material point

```
 DIMENSION STRESS(*),DDSDDE(NTENS,*),STRAN(*),DSTRAN(*),
*       DFGRD(3,3),PREDEF(NPREDF),DPREDEF(NPREDF),COORDS(3)
     ...

 CALL MATERIAL_LIB_MECH(MATERIALLIB,STRESS,DDSDDE,STRAN,DSTRAN,
*      NPT,DVDV0,DVMAT,DFGRD,PREDEF,DPREDEF,NPREDF,CELENT,COORDS)
     ...
```

MATERIAL_LIB_MECH

Variables to Be Provided to the Utility Routine

MATERIALLIB Variable containing information about the Abaqus material. This variable is passed into user subroutine UELMAT

STRAN Strain at the beginning of the increment

DSTRAN Strain increment

NPT Integration point number

DVDV0 Ratio of the current volume to the reference volume at the integration point

DVMAT Volume at the integration point

MATERIAL_LIB_MECH

Variables to Be Provided to the Utility Routine

DFGRD	Array containing the deformation gradient at the end of the increment
PREDEF	Array of interpolated values of predefined field variables at the integration point at the start of the increment
DPREDEF	Array of increments of predefined field variables
NPREDF	Number of predefined field variables, including temperature
CELENT	Characteristic element length
COORDS	An array containing the coordinates of this point

MATERIAL_LIB_MECH

Variables Returned from the Utility Routine

STRESS Stress tensor at the end of the increment

DDSDDE Jacobian matrix of the constitutive model

DDSDDE(i, j) defines the change in the i^{th} stress component at the end of the time increment caused by an infinitesimal perturbation of the j^{th} component of the strain increment array

MATERIAL_LIB_HT

Returns **heat fluxes**, **internal energy time derivative**, **volumetric heat generation rate**, and **their derivatives** at the element material point

```
 DIMENSION PREDEF(NPREDEF),DPREDEF(NPREDEF),DTEMDX(*),
*        RHODUDG(*),FLUX(*),DFDT(*),DFDG(NDIM,*),DRPLDT(*),
*        COORDS(3)
     ...

 CALL MATERIAL_LIB_HT(MATERIALLIB,RHOUDOT,RHODUDT,RHODUDG,
*        FLUX,DFDT,DFDG,RPL,DRPLDT,NPT,DVMAT,PREDEF,
*        DPREDEF,NPREDF,TEMP,DTEMP,DTEMDX,CELENT,COORDS)
     ...
```

MATERIAL_LIB_HT

Variables to Be Provided to the Utility Routine

MATERIALLIB Variable containing information about the Abaqus material. This variable is passed into user subroutine UELMAT

NPT Integration point number

DVMAT Volume at the integration point

PREDEF Array of interpolated values of predefined field variables at the integration point at the start of the increment

DPREDEF Array of increments of predefined field variables

NPREDF Number of predefined field variables, including temperature

MATERIAL_LIB_HT

Variables to Be Provided to the Utility Routine

TEMP	Temperature at the integration point at the start of the increment, θ
DTEMDX	Spatial gradients of temperature, $\partial\theta/\partial x$ the end of the increment
CELENT	Characteristic element length
COORDS	An array containing the coordinates of this point

MATERIAL_LIB_HT

Variables Returned from the Utility Routine

RHOUDOT $\rho \frac{dU}{dt}$

Time derivative of the internal thermal energy per unit mass, U, multiplied by density at the end of increment

RHODUDT $\rho \frac{\partial U}{\partial \theta}$

Variation of internal thermal energy per unit mass with respect to temperature multiplied by density evaluated at the end of the increment

RHODUDG $\rho \frac{\partial U}{\partial \left(\frac{\partial \theta}{\partial x}\right)}$

Variation of internal thermal energy per unit mass with respect to the spatial gradients of temperature multiplied by density at the end of the increment

FLUX

Heat flux vector, $\boldsymbol{f}$, at the end of the increment

MATERIAL_LIB_HT

Variables Returned from the Utility Routine

DFDT Variation of the heat flux vector with respect to temperature evaluated at the end of the increment

$$\frac{\partial \boldsymbol{f}}{\partial \theta}$$

DFDG Variation of the heat flux vector with respect to the spatial gradients of temperature at the end of the increment

$$\frac{\partial \boldsymbol{f}}{\partial \left(\frac{\partial \theta}{\partial x}\right)}$$

RPL Volumetric heat generation per unit time at the end of the increment

DRPLDT Variation of RPL with respect to temperature

User-defined Element

When user-defined elements is useful ?

- Modeling **nonstructural** physical processes that are coupled to structural behavior
- Applying solution-dependent loads
- Modeling active control mechanisms

Advantages of the User-defined element Subroutine instead of writing a complete FEA code

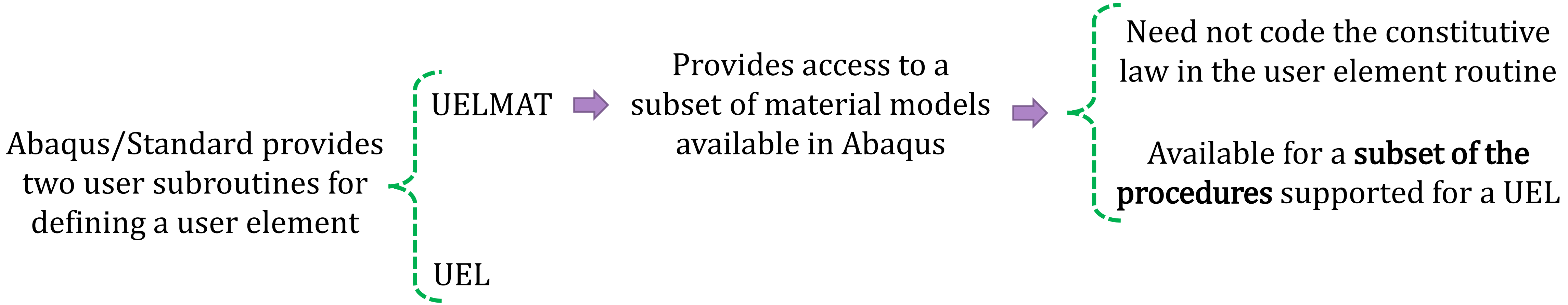
- ABAQUS offers a large selection of structural elements, analysis procedures, and modeling tools
- ABAQUS offers pre- and postprocessing
- Maintaining and porting subroutines is much easier than maintaining and porting a complete finite element program

User-defined Element

Multiple user elements can be implemented in a single UEL/UELMAT/VUEL routine and can be utilized together

A linear user element can be created in Abaqus/Standard by defining the stiffness and mass matrices directly using the *MATRIX option

A nonlinear finite element is implemented in user subroutine UEL (Abaqus/Standard), UELMAT (Abaqus/Standard), or VUEL (Abaqus/Explicit)



Abaqus/Standard provides two user subroutines for defining a user element

UELMAT

Provides access to a subset of material models available in Abaqus

Need not code the constitutive law in the user element routine

Available for a **subset of the procedures** supported for a UEL

UEL

User-defined Element

Characteristics of the User element

- The number of nodes on the element
- The number of coordinates present at each node
- The degrees of freedom active at each node

Element Properties must be determined

- The number of element properties to be defined external to the UEL
- The number of solution-dependent state variables (SDVs) to be stored per element
- The number of (distributed) load types available for the element

User-defined Element

F_{ext}^{N} is the external flux (due to applied distributed loads) and F_{int}^{N} is the internal flux (due to stresses, e.g.) at node N

$$F^{N} = F_{ext}^{N} - F_{int}^{N} = 0 \longrightarrow AMTRIX = RHS$$

LHS

RHS

Degrees of Freedom {

- Displacements → Nodal Forces
- Rotations → Moments
- Temperatures → Heat Fluxes

In nonlinear user elements the fluxes/forces will often depend on the increments in the degrees of freedom Δu^{N} and the $\underbrace{\text{internal state variables } H^{\alpha}}$

State variables must be updated in the user subroutine

User-defined Element

The solution of the (nonlinear) system of equations in **general steps** requires defining the **element Jacobian** (stiffness matrix):

Element Jacobian / Stiffness Matrix ●
AMATRIX ●

$$K^{NM} = -\frac{dF^N}{du^M}$$

The Jacobian should include all direct and indirect dependencies of F^N on u^N, which includes terms of the form

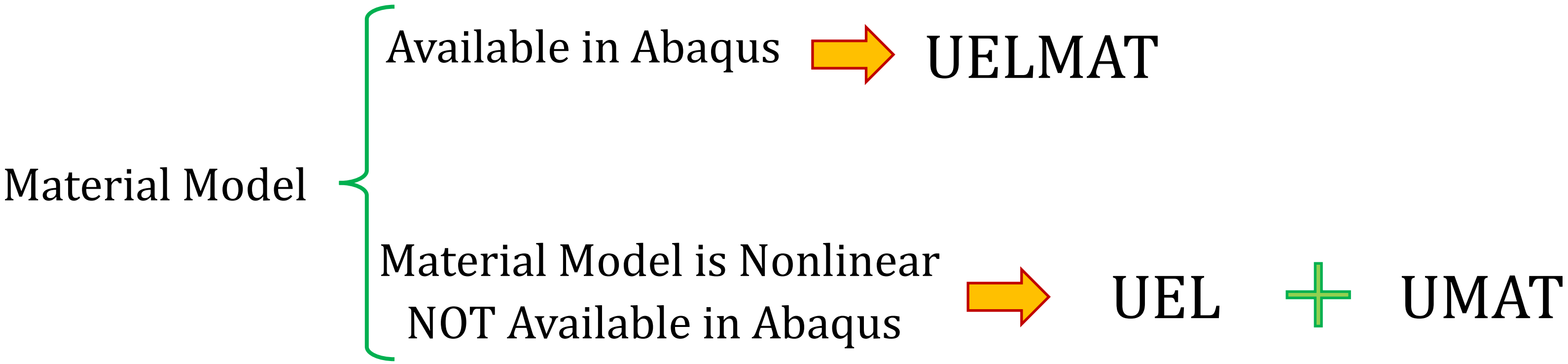
$$K^{NM} = -\frac{\partial F^N}{\partial H^\alpha}\frac{\partial H^\alpha}{\partial u^M}$$

→ Internal State Variables

Element Jacobian

- A more accurately defined Jacobian improves convergence in general steps
- The Jacobian (stiffness) determines the solution for linear perturbation steps, so it must be exact
- The Jacobian can be symmetric or nonsymmetric

User-defined Element



Material Model

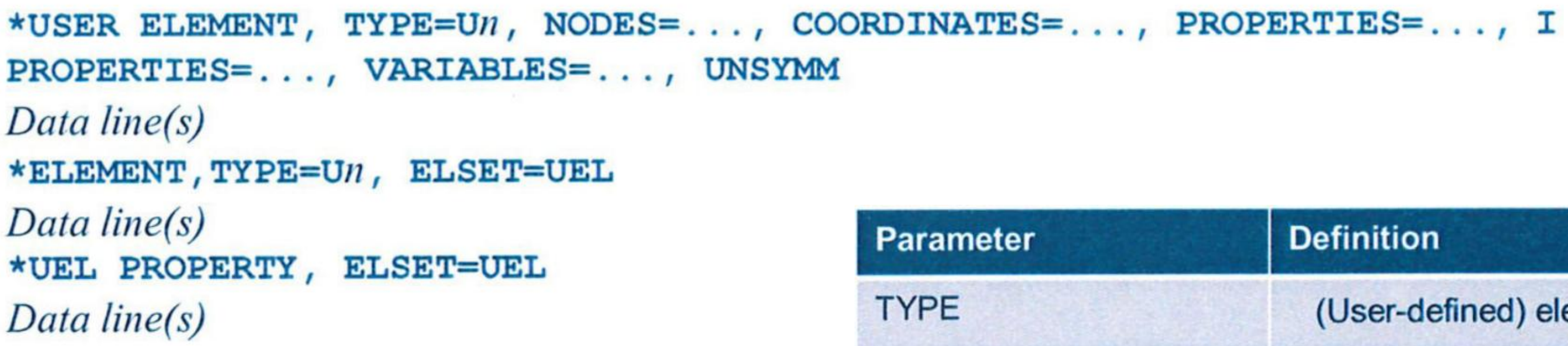
Available in Abaqus → UELMAT

Material Model is Nonlinear
NOT Available in Abaqus → UEL + UMAT

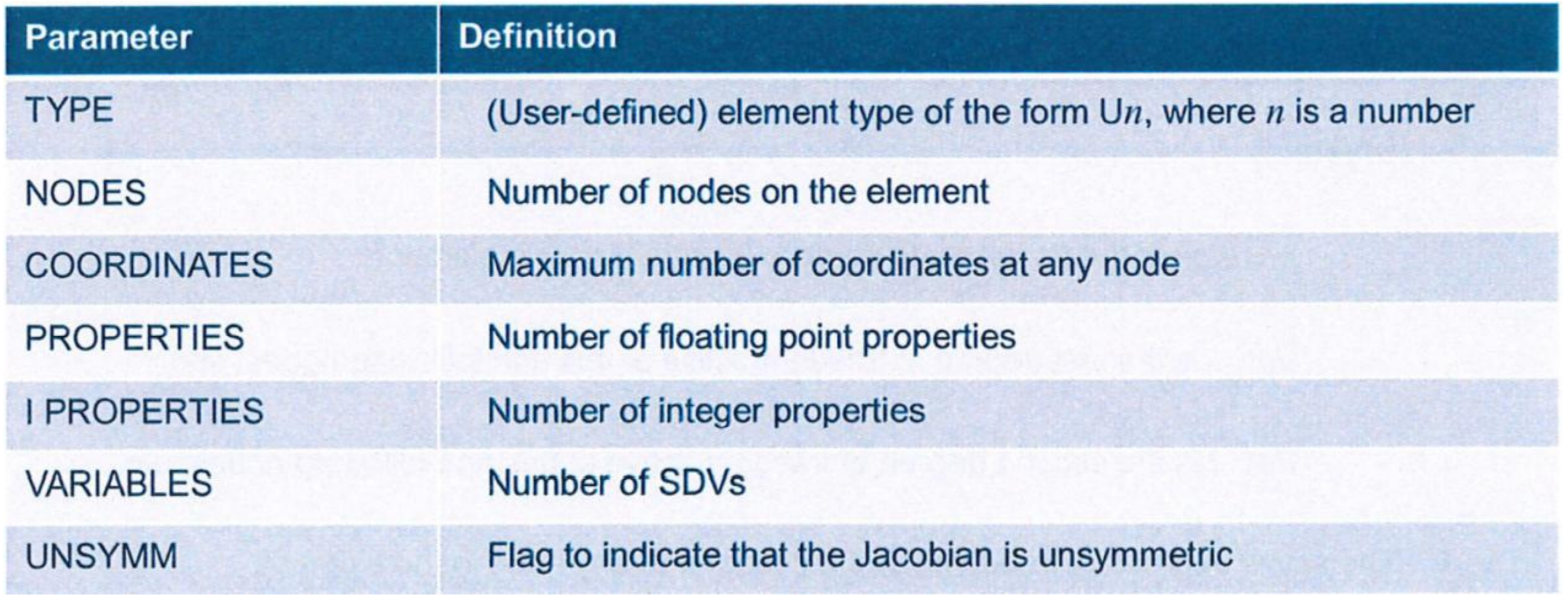
User-defined Element

Writing INP

- A user element is defined with the *USER ELEMENT option
- This option must appear in the input file before the user element is invoked with the *ELEMENT option
- The syntax for interfacing to UEL is as follows:



```
*USER ELEMENT, TYPE=Un, NODES=..., COORDINATES=..., PROPERTIES=..., I
PROPERTIES=..., VARIABLES=..., UNSYMM
Data line(s)
*ELEMENT,TYPE=Un, ELSET=UEL
Data line(s)
*UEL PROPERTY, ELSET=UEL
Data line(s)
```



Parameter	Definition
TYPE	(User-defined) element type of the form U*n*, where *n* is a number
NODES	Number of nodes on the element
COORDINATES	Maximum number of coordinates at any node
PROPERTIES	Number of floating point properties
I PROPERTIES	Number of integer properties
VARIABLES	Number of SDVs
UNSYMM	Flag to indicate that the Jacobian is unsymmetric

User-defined Element

```
*USER ELEMENT, TYPE=U1, NODES=2, PROPERTIES=4, I PROPERTIES=2
COORDINATES=3, VARIABLES=12, UNSYMM
1, 2, 3
```

Data line(s)

```
*ELEMENT, TYPE=U1
101, 101, 102
```

Data line(s)

```
*ELGEN, ELSET=UEL
101, 5
```

Data line(s)

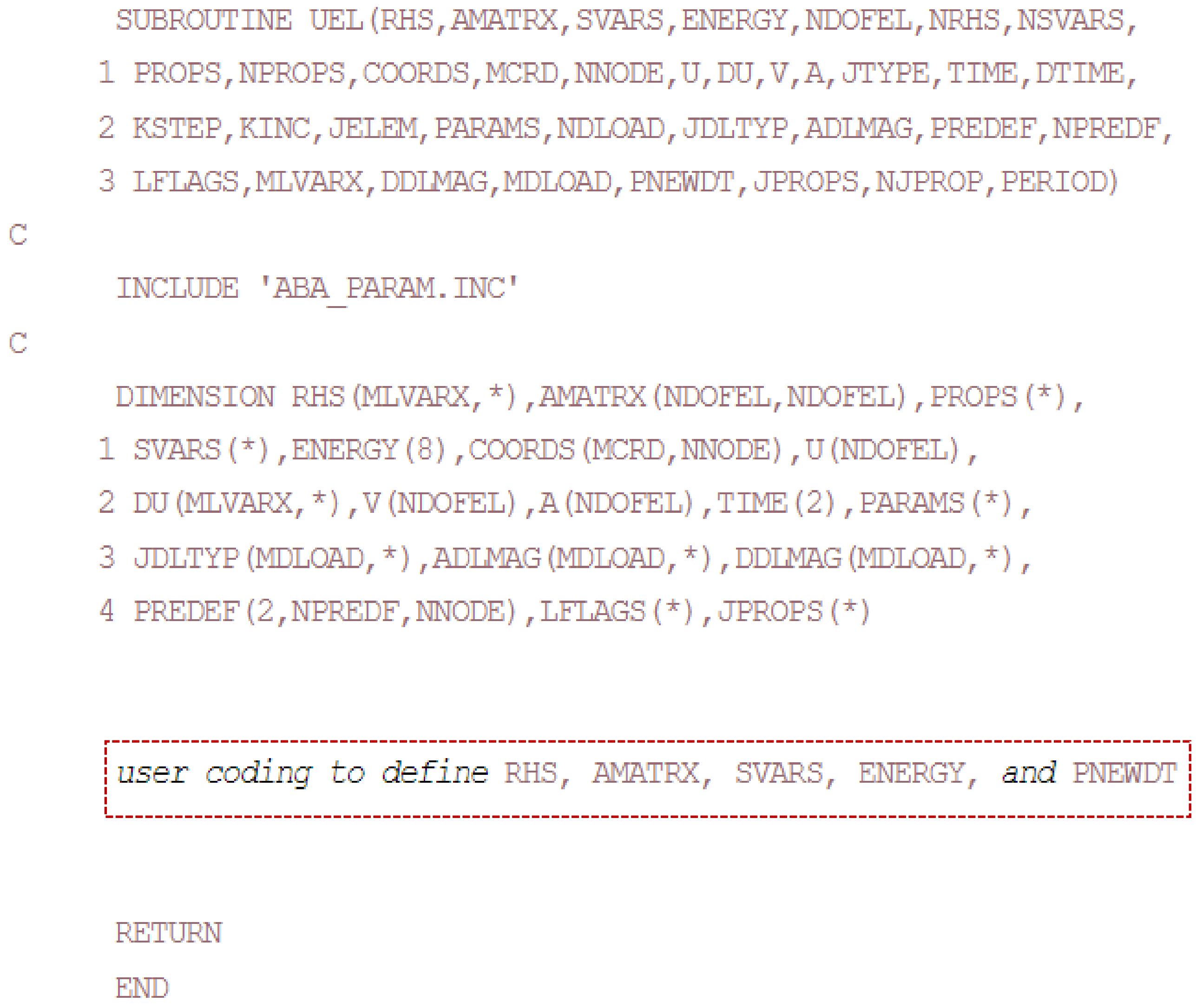
```
*UEL PROPERTY, ELSET=UEL
0.002, 2.1E11, 0.3, 7200., 2,5
```

Data line(s)

Enter the values of the element properties.
Enter all floating-point values first, followed immediately by the integer values

UEL

Abaqus User Subroutine To Define An (Nonlinear) Element



```
      SUBROUTINE UEL(RHS,AMATRX,SVARS,ENERGY,NDOFEL,NRHS,NSVARS,
     1 PROPS,NPROPS,COORDS,MCRD,NNODE,U,DU,V,A,JTYPE,TIME,DTIME,
     2 KSTEP,KINC,JELEM,PARAMS,NDLOAD,JDLTYP,ADLMAG,PREDEF,NPREDF,
     3 LFLAGS,MLVARX,DDLMAG,MDLOAD,PNEWDT,JPROPS,NJPROP,PERIOD)
C
      INCLUDE 'ABA_PARAM.INC'
C
      DIMENSION RHS(MLVARX,*),AMATRX(NDOFEL,NDOFEL),PROPS(*),
     1 SVARS(*),ENERGY(8),COORDS(MCRD,NNODE),U(NDOFEL),
     2 DU(MLVARX,*),V(NDOFEL),A(NDOFEL),TIME(2),PARAMS(*),
     3 JDLTYP(MDLOAD,*),ADLMAG(MDLOAD,*),DDLMAG(MDLOAD,*),
     4 PREDEF(2,NPREDF,NNODE),LFLAGS(*),JPROPS(*)


      user coding to define RHS, AMATRX, SVARS, ENERGY, and PNEWDT


      RETURN
      END
```

Variables Passed in for Information

DTIME → Time increment

PERIOD → Time period of the current step

NDOFEL → Number of degrees of freedom in the element

MLVARX → Dimensioning parameter used when **several displacement** or **right-hand-side** vectors are used

```
RHS(MLVARX,*), DU(MLVARX,*)
```

NRHS → Number of load vectors

- NRHS=1 in most nonlinear problems
- NRSH=2 for the modified Riks static procedure
- Greater than 1 in some linear analysis procedures and during substructure generation

↓

For example, in the recovery path for the **direct steady-state** procedure, it is 2 to accommodate the **real** and **imaginary** parts of the vectors

Variables Passed in for Information

`NSVARS` → User-defined **number of solution-dependent state variables** associated with the element

`NPROPS` → User-defined **number of real property** values associated with the element

`NJPROP` → User-defined **number of integer property** values associated with the element

`MCRD <= 3` → Number of Coordinate Components → The maximum of
- Maximum number of coordinates required at any node point
- Value of the largest active degree of freedom

`NNODE` → User-defined **number of nodes on the element**

Variables Passed in for Information

User element type ID

JTYPE ---> Integer defining the element type(n)

Abaqus/Standard	Un	($n \leq 1000$)
Abaqus/Explicit	VUn	($n \leq 10000$)

KSTEP ---> Current step number

KINC ---> Current increment number

JELEM ---> User-assigned element number

NDLOAD ---> Identification number of the distributed load or flux currently **active** on this element

MDLOAD ---> Total number of distributed loads and/or fluxes defined on this element

NPREDF ---> Number of predefined field variables, including temperature
For user elements Abaqus/Standard uses one value for each field variable per node

Variables Passed in for Information

`PROPS(*)` → A **floating-point** array containing the NPROPS real property values defined for use with this element. NPROPS is the user-specified number of real property values

`JPROPS(*)` → An **integer** array containing the NJPROP **integer** property values defined for use with this element. NJPROP is the user-specified number of integer property values

`COORDS(MCRD, NNODE)` → An array containing the **original coordinates** of the nodes of the element COORDS(K1,K2) is the $K1^{th}$ coordinate of the $K2^{th}$ node of the element

`JDLTYP(*)` → An array containing the integers used to define distributed load types for the element

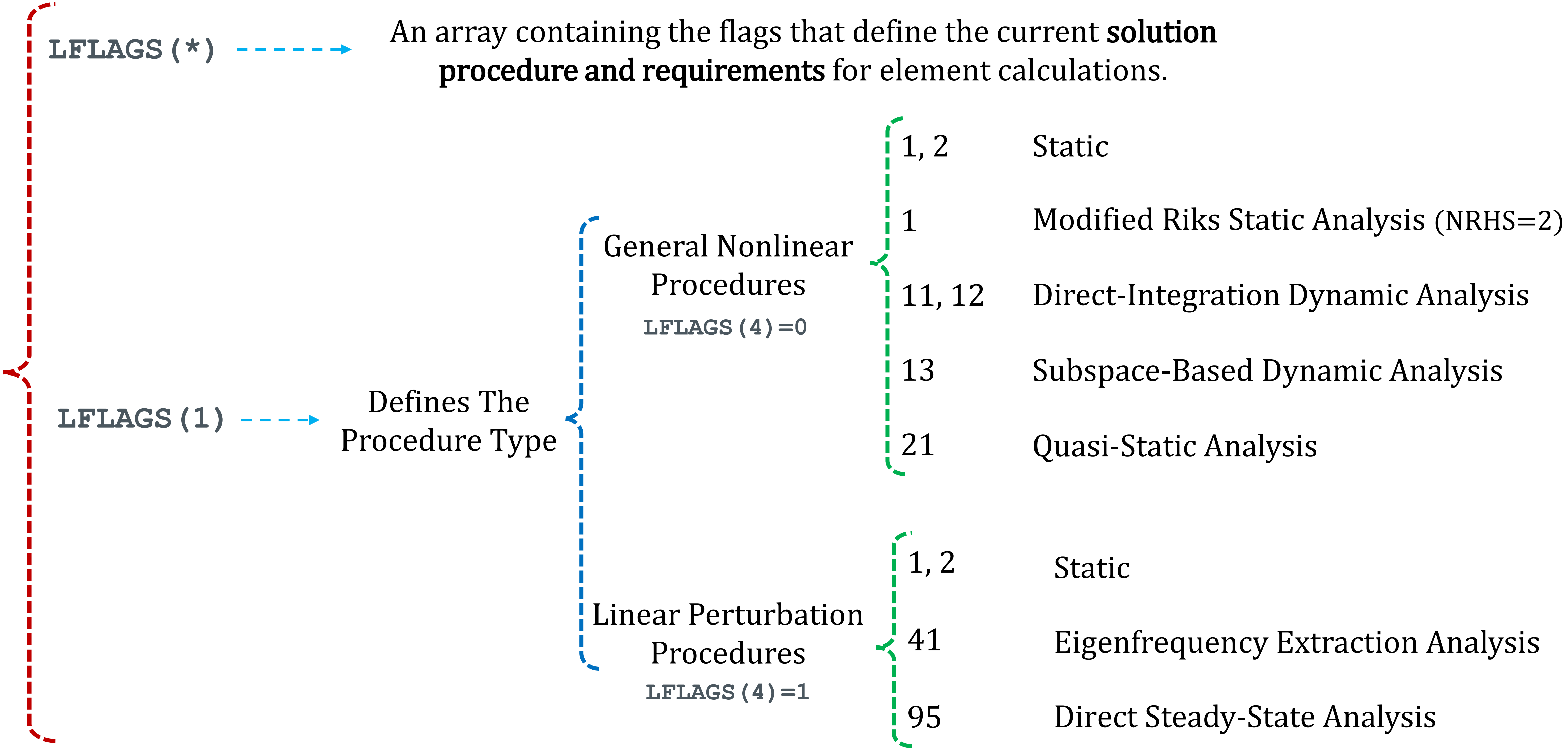
Loads of type Un are identified by the integer value n in JDLTYP

Loads of type UnNU are identified by the negative integer value −n in JDLTYP

JDLTYP(K1,K2) is the identifier of the $K1^{th}$ distributed load in the $K2^{th}$ load case

For general nonlinear steps: K2 =1

Variables Passed in for Information



LFLAGS(*) → An array containing the flags that define the current **solution procedure and requirements** for element calculations.

LFLAGS(1) → Defines The Procedure Type

General Nonlinear Procedures

LFLAGS(4)=0

1, 2	Static
1	Modified Riks Static Analysis (NRHS=2)
11, 12	Direct-Integration Dynamic Analysis
13	Subspace-Based Dynamic Analysis
21	Quasi-Static Analysis

Linear Perturbation Procedures

LFLAGS(4)=1

1, 2	Static
41	Eigenfrequency Extraction Analysis
95	Direct Steady-State Analysis

Variables Passed in for Information

LFLAGS (1)	Procedure	Comments
1, 2	Static	Automatic/fixed time incrementation
11,12	Dynamic	Automatic/fixed time incrementation
21,22	Visco	Quasi-static; explicit/implicit time integration
31	Heat Transfer	Steady-state
32, 33	Heat Transfer	Transient; fixed/automatic time incrementation
41	Frequency extraction	
61	Geostatic	
62, 63	Soils	Steady-state; fixed/automatic time incrementation
64, 65	Soils	Transient; fixed/automatic time incrementation
71	Coupled thermal-stress	Steady-state
72,73	Coupled thermal-stress	Transient; fixed/automatic time incrementation
75	Coupled thermal-electrical	Steady-state
76,77	Coupled thermal-electrical	Transient; fixed/automatic time incrementation

Variables Passed in for Information

`LFLAGS(*)` → An array containing the flags that define the current **solution procedure and requirements** for element calculations.

`LFLAGS(2)=`

- 0 Small-displacement analysis
- 1 Large-displacement analysis (nonlinear geometric effects included in the step)

Variables Passed in for Information

LFLAGS(3)=

1 Normal implicit time incrementation procedure. User subroutine UEL must define the residual vector in RHS and the Jacobian matrix in AMATRX.

2 Define the current stiffness matrix (AMATRX = $K^{NM} = -\frac{\partial F^N}{\partial u^M}$ or $-\frac{\partial G^N}{\partial u^M}$) only

3 Define the current damping matrix (AMATRX = $C^{NM} = -\frac{\partial F^N}{\partial \dot{u}^{\mathrm{M}}}$ or $-\frac{\partial G^N}{\partial \dot{u}^{\mathrm{M}}}$) only

4 Define the current mass matrix (AMATRX = $M^{NM} = -\frac{\partial F^N}{\partial \ddot{u}^M}$) only.

Abaqus/Standard always requests an initial mass matrix at the start of the analysis.

5 Define the **current residual or load vector** (RHS =F^N) only

6 Define the current **mass matrix** and the **residual vector** for the initial acceleration calculation (or the calculation of accelerations after impact)

100 Define perturbation quantities for output.

Not available for direct steady-state dynamic and mode-based procedures

Variables Passed in for Information

`LFLAGS(4)=`

- 0 The step is a general step
- 1 The step is a linear perturbation step

`LFLAGS(5)=`

- 0 The current approximations to u^M, etc. were based on **Newton corrections**
- 1 The current approximations were found by **extrapolation** from the previous increment

`LFLAGS(7)=`

- 1 When the damping matrix flag is set, the **viscous damping** matrix is defined
- 2 When the damping matrix flag is set, the **structural damping** matrix is defined

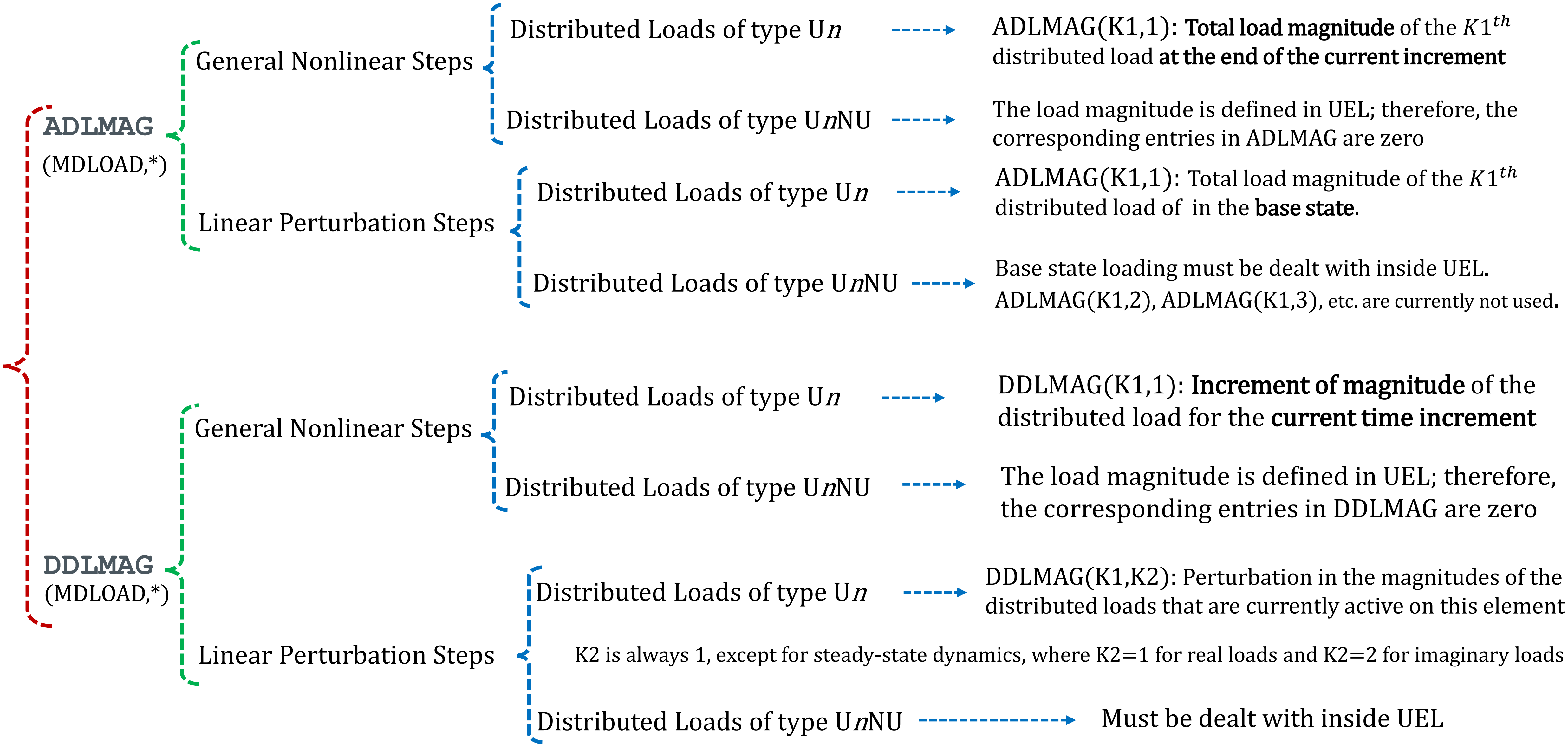
Variables Passed in for Information

```
U, V, A (NDOFEL)
DU(MLVARX,*)
```

Arrays containing the current estimates of the **basic solution variables** (displacements, rotations, temperatures, depending on the degree of freedom) at the nodes of the element at the **end of the current incremen**t. Values are provided as follows:

$U\ (K1\ )$	Total values of the variables. If this is a linear perturbation step, it is the value in the **base state**.
$DU\ (K1, KRHS)$	**Incremental values of the variables for the current increment for right-hand-side KRHS.** For eigenvalue extraction step, this is the eigenvector magnitude for eigenvector KRHS. For steady-state dynamics, KRHS = 1 denotes real components of perturbation displacement and KRHS = 2 denotes imaginary components of perturbation displacement.
$V\ (K1\ )$	Time rate of change of the variables (velocities, rates of rotation). Defined for implicit dynamics only (LFLAGS (1) = 11 or 12).
$A\ (K1\ )$	Accelerations of the variables. Defined for implicit dynamics only (LFLAGS (1) = 11 or 12).

Variables Passed in for Information



- **ADLMAG** (MDLOAD,*)
  - General Nonlinear Steps
    - Distributed Loads of type Un → ADLMAG(K1,1): **Total load magnitude** of the $K1^{th}$ distributed load **at the end of the current increment**
    - Distributed Loads of type UnNU → The load magnitude is defined in UEL; therefore, the corresponding entries in ADLMAG are zero
  - Linear Perturbation Steps
    - Distributed Loads of type Un → ADLMAG(K1,1): Total load magnitude of the $K1^{th}$ distributed load of in the **base state.**
    - Distributed Loads of type UnNU → Base state loading must be dealt with inside UEL. ADLMAG(K1,2), ADLMAG(K1,3), etc. are currently not used.
- **DDLMAG** (MDLOAD,*)
  - General Nonlinear Steps
    - Distributed Loads of type Un → DDLMAG(K1,1): **Increment of magnitude** of the distributed load for the **current time increment**
    - Distributed Loads of type UnNU → The load magnitude is defined in UEL; therefore, the corresponding entries in DDLMAG are zero
  - Linear Perturbation Steps
    - Distributed Loads of type Un → DDLMAG(K1,K2): Perturbation in the magnitudes of the distributed loads that are currently active on this element
    - K2 is always 1, except for steady-state dynamics, where K2=1 for real loads and K2=2 for imaginary loads
    - Distributed Loads of type UnNU → Must be dealt with inside UEL

Variables Passed in for Information

`PREDEF(2,NPREDF,NNODE)`

An array containing the values of predefined field variables, such as temperature in an uncoupled stress/displacement analysis, at the nodes of the element

Index Of The Array

- First (K1)
  - 1 The value of the field variable at the **end of the increment**
  - 2 The **increment in the field variable**
- Second (K2)
  - 1 The temperature
  - 2, … The predefined field variables
- Third (K3) The local node number on the element

In cases where temperature is not defined, the predefined field variables begin with index 1

PREDEF (K1,1,K3)	Temperature.
PREDEF (K1,2, ,K3)	First predefined field variable.
PREDEF (K1,3, K3)	Second predefined field variable.
Etc.	Any other predefined field variable.
PREDEF (K1,K2, K3)	Total or incremental value of the $K2^{th}$ predefined field variable at the $K3^{th}$ node of the element.
PREDEF (1,K2,K3)	Values of the variables at the end of the current increment.
PREDEF (2,K2,K3)	Incremental values corresponding to the current time increment.

Variables Passed in for Information

PARAMS(*)

An array containing the parameters associated with the **solution procedure**. The entries in this array depend on the solution procedure currently being used when UEL is called, as indicated by the entries in the LFLAGS array.

For implicit dynamics (LFLAGS(1) = 11 or 12) PARAMS contains the **integration operator values**, as:

PARAMS

- PARAMS(1) → α
- PARAMS(2) → β
- PARAMS(3) → γ

TIME(1) Current value of step time or frequency

TIME(2) Current value of total time

Variables to Be Defined

These arrays depend on the value of the **LFLAGS** array

RHS(MLVARX,*) → An array containing the contributions of this element to the right-hand-side vectors of the overall system of equations

AMATRX(NDOFEL,NDOFEL) → An array containing the contribution of this element to the Jacobian (stiffness) or other matrix of the overall system of equations

At time Increment n+1

$$\overset{\text{Residual}}{\mathbf{R}}\left(\mathbf{d}^{\mathbf{n+1}}, t^{n+1}\right) = \mathbf{F}_{ext}\left(\mathbf{d}^{\mathbf{n+1}}, t^{n+1}\right) - \mathbf{F}_{int}\left(\mathbf{d}^{\mathbf{n+1}}, t^{n+1}\right) = 0$$

Linearized Model Of The Nonlinear Equations

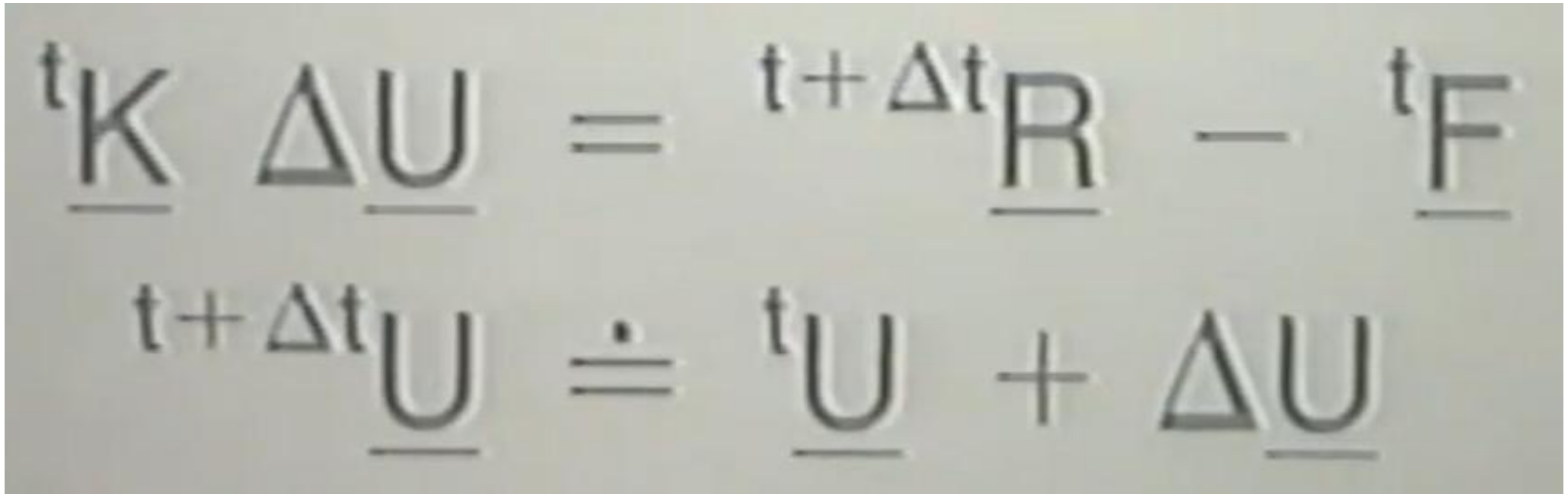
At time Increment n+1
At Iteration m

$$\mathbf{R}(\mathbf{d}_{m+1}, t^{n+1}) = \mathbf{R}(\mathbf{d}_m, t^{n+1}) + \underbrace{\frac{\partial \mathbf{R}(\mathbf{d}_m, t^{n+1})}{\partial \mathbf{d}}}_{\text{Jacobian Matrix}} \underbrace{(\mathbf{d}_{\mathrm{m+1}} - \mathbf{d}_{\mathrm{m}})}_{\Delta \mathbf{d}} = 0$$

$$\underbrace{-\frac{\partial \mathbf{R}(\mathbf{d}_m, t^{n+1})}{\partial \mathbf{d}}}_{\text{NDOFEL} \times \text{NDOFEL}} \Delta \mathbf{d} = \underbrace{\mathbf{R}(\mathbf{d}_m, t^{n+1})}_{\text{DU(MLVARX,*)}}$$

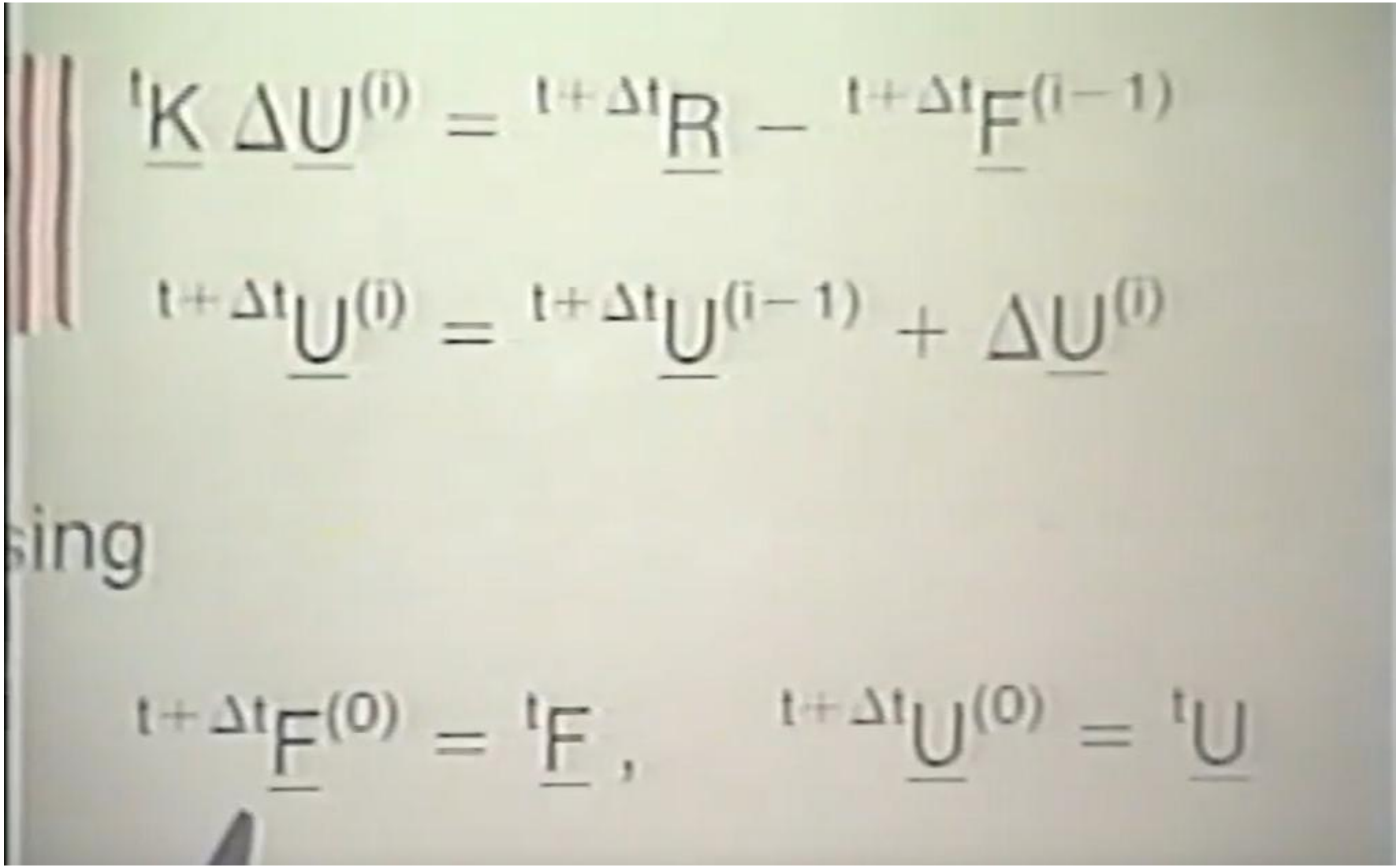
AMATRX

RHS



$$ {}^{t}\underline{K}\;\Delta\underline{U} = {}^{t+\Delta t}\underline{R} - {}^{t}\underline{F} $$

$$ {}^{t+\Delta t}\underline{U} \doteq {}^{t}\underline{U} + \Delta\underline{U} $$

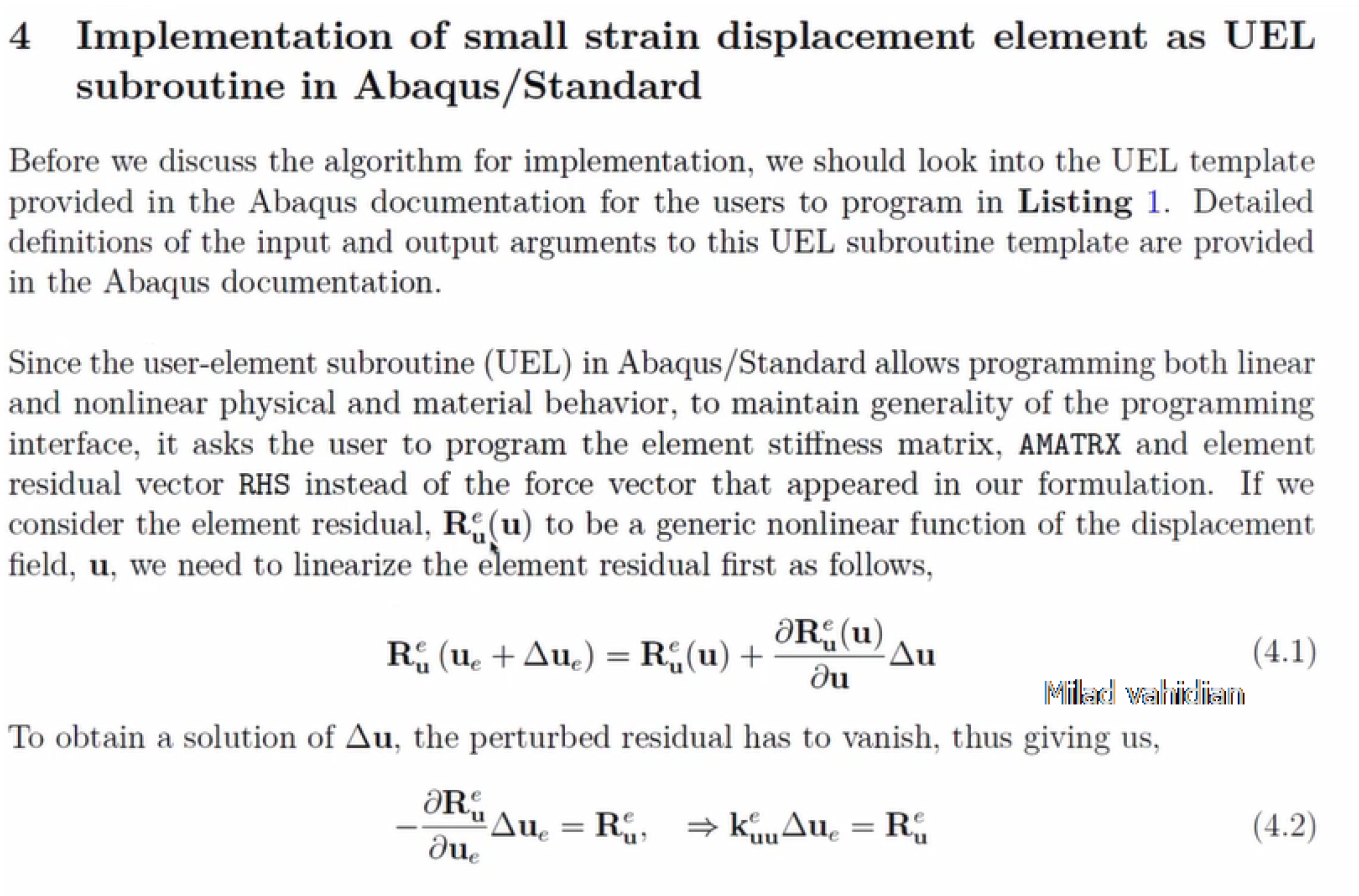


$$ {}^{t}\underline{K}\,\Delta\underline{U}^{(i)} = {}^{t+\Delta t}\underline{R} - {}^{t+\Delta t}\underline{F}^{(i-1)} $$

$$ {}^{t+\Delta t}\underline{U}^{(i)} = {}^{t+\Delta t}\underline{U}^{(i-1)} + \Delta\underline{U}^{(i)} $$

sing

$$ {}^{t+\Delta t}\underline{F}^{(0)} = {}^{t}\underline{F}, \qquad {}^{t+\Delta t}\underline{U}^{(0)} = {}^{t}\underline{U} $$



4 Implementation of small strain displacement element as UEL subroutine in Abaqus/Standard

Before we discuss the algorithm for implementation, we should look into the UEL template provided in the Abaqus documentation for the users to program in **Listing** 1. Detailed definitions of the input and output arguments to this UEL subroutine template are provided in the Abaqus documentation.

Since the user-element subroutine (UEL) in Abaqus/Standard allows programming both linear and nonlinear physical and material behavior, to maintain generality of the programming interface, it asks the user to program the element stiffness matrix, `AMATRX` and element residual vector `RHS` instead of the force vector that appeared in our formulation. If we consider the element residual, $\mathbf{R}^e_{\mathbf{u}}(\mathbf{u})$ to be a generic nonlinear function of the displacement field, $\mathbf{u}$, we need to linearize the element residual first as follows,

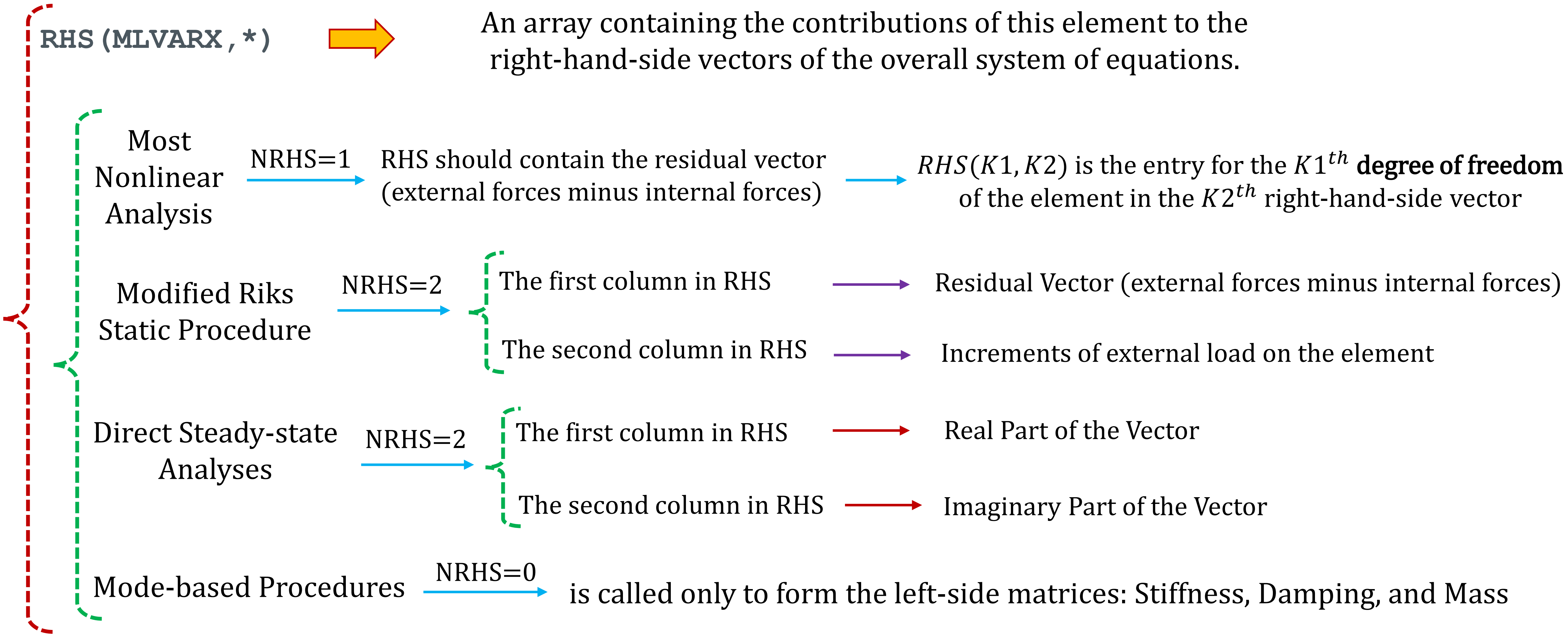
$$\mathbf{R}^e_{\mathbf{u}}\left(\mathbf{u}_e + \Delta\mathbf{u}_e\right) = \mathbf{R}^e_{\mathbf{u}}(\mathbf{u}) + \frac{\partial \mathbf{R}^e_{\mathbf{u}}(\mathbf{u})}{\partial \mathbf{u}}\Delta\mathbf{u} \qquad (4.1)$$

To obtain a solution of $\Delta\mathbf{u}$, the perturbed residual has to vanish, thus giving us,

$$-\frac{\partial \mathbf{R}^e_{\mathbf{u}}}{\partial \mathbf{u}_e}\Delta\mathbf{u}_e = \mathbf{R}^e_{\mathbf{u}}, \quad \Rightarrow \mathbf{k}^e_{\mathbf{uu}}\Delta\mathbf{u}_e = \mathbf{R}^e_{\mathbf{u}} \qquad (4.2)$$

Variables to Be Defined

These arrays depend on the value of the **LFLAGS** array



`RHS(MLVARX,*)` → An array containing the contributions of this element to the right-hand-side vectors of the overall system of equations.

Most Nonlinear Analysis → NRHS=1 → RHS should contain the residual vector (external forces minus internal forces) → $RHS(K1, K2)$ is the entry for the $K1^{th}$ **degree of freedom** of the element in the $K2^{th}$ right-hand-side vector

Modified Riks Static Procedure → NRHS=2 →
- The first column in RHS → Residual Vector (external forces minus internal forces)
- The second column in RHS → Increments of external load on the element

Direct Steady-state Analyses → NRHS=2 →
- The first column in RHS → Real Part of the Vector
- The second column in RHS → Imaginary Part of the Vector

Mode-based Procedures → NRHS=0 → is called only to form the left-side matrices: Stiffness, Damping, and Mass

Variables to Be Defined

These arrays depend on the value of the **LFLAGS** array

`AMATRX(NDOFEL,NDOFEL)` → An array containing the contribution of this element to the Jacobian (stiffness) or other matrix of the overall system of equations

The particular matrix required at any time depends on the entries in the LFLAGS array

All nonzero entries in AMATRX should be defined, even if the matrix is symmetric

- The matrix is unsymmetric → AMATRX
- The matrix is symmetric → AMATRX $= \frac{1}{2}([A] + [A]^T)$

Variables to Be Defined

These arrays depend on the value of the **LFLAGS** array

`SVARS(*)` → **An array** containing the values of the **solution-dependent state variables** associated with this element

The number of such variables is **NSVARS**

General Nonlinear Steps → This array is passed into UEL containing the values of these variables at the start of the current increment. They should be updated to be the values at the end of the increment, unless the procedure during which UEL is being called does not require such an update.

Linear Perturbation Steps → This array is passed into UEL containing the values of these variables in the **base state**. They should be returned containing perturbation values if you want to output such quantities.

When KINC is equal to zero, the call to UEL is made for zero increment output.
In this case the values returned will be used only for output purposes and are not updated permanently.

Variables to Be Defined

These arrays depend on the value of the **LFLAGS** array

`ENERGY(8)`

General Nonlinear Steps → ENERGY contains the values of the energy quantities associated with the element

⇩

The values in this array when UEL is called are the element energy quantities at the start of the current increment. They should be updated to the values at the end of the **current increment**

Linear Perturbation Steps → ENERGY contains the values of the energy in the **base state**

⇩

They should be returned containing perturbation values if you wish to output such quantities

Mode-based Procedures ⇨ They are not available for updates

Variables to Be Defined

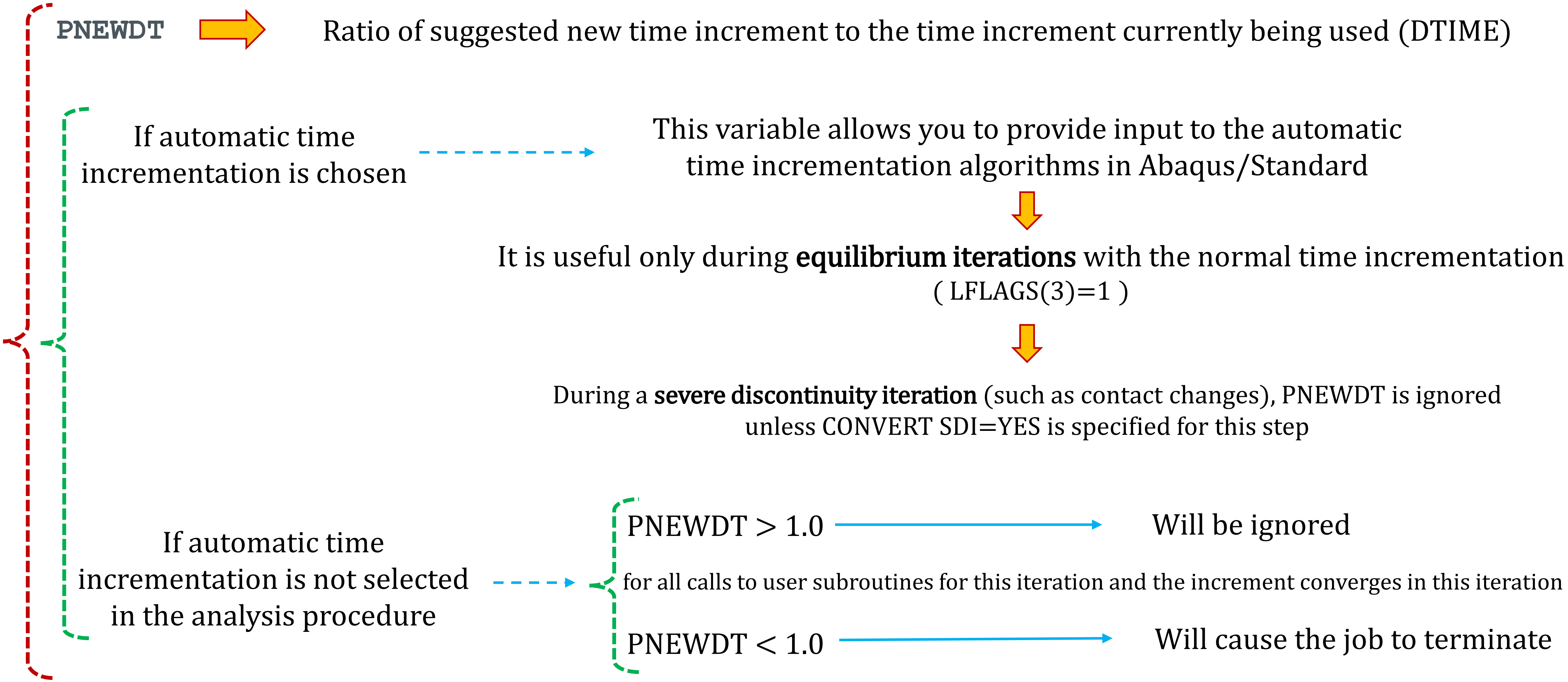
These arrays depend on the value of the **LFLAGS** array

`ENERGY(1)`	Kinetic energy
`ENERGY(2)`	Elastic strain energy
`ENERGY(3)`	Creep dissipation
`ENERGY(4)`	Plastic dissipation
`ENERGY(5)`	Viscous dissipation
`ENERGY(6)`	“Artificial strain energy”
`ENERGY(7)`	Electrostatic energy
`ENERGY(8)`	Incremental work done by loads applied within the user element

When KINC is equal to zero, the call to UEL is made for zero increment output. In this case the energy values returned will be used only for output purposes and are not updated permanently.

Associated with such effects as artificial stiffness introduced to control hourglassing or other singular modes in the element.

Variables That Can Be Updated



PNEWDT → Ratio of suggested new time increment to the time increment currently being used (DTIME)

If automatic time incrementation is chosen → This variable allows you to provide input to the automatic time incrementation algorithms in Abaqus/Standard

It is useful only during **equilibrium iterations** with the normal time incrementation (LFLAGS(3)=1)

During a **severe discontinuity iteration** (such as contact changes), PNEWDT is ignored unless CONVERT SDI=YES is specified for this step

If automatic time incrementation is not selected in the analysis procedure →

- $PNEWDT > 1.0$ → Will be ignored
- for all calls to user subroutines for this iteration and the increment converges in this iteration
- $PNEWDT < 1.0$ → Will cause the job to terminate

Variables That Can Be Updated

If Automatic Time Incrementation Is Chosen:

If PNEWDT is redefined to be less than 1.0

Abaqus/Standard **must** abandon the time increment and attempt it again with a smaller time increment. The suggested new time increment provided to the automatic time integration algorithms is PNEWDT × DTIME, where the PNEWDT used is the minimum value for all calls to user subroutines that allow redefinition of PNEWDT for this iteration

If PNEWDT is given a value that is greater than 1.0

(For all calls to user subroutines for this iteration and the increment converges in this iteration)

Abaqus/Standard **may** increase the time increment. The suggested new time increment provided to the automatic time integration algorithms is PNEWDT × DTIME, where the PNEWDT used is the minimum value for all calls to user subroutines for this iteration.

Hints to Write UEL

UEL Variables

Available in UEL

- Coordinates; displacements; incremental displacements; and, for dynamics, velocities and accelerations
- SDVs at the start of the increment
- Total and incremental values of time, temperature, and user-defined field variables
- User element properties
- Load types as well as total and incremental load magnitudes
- Element type and user-defined element number
- Procedure type flag and, for dynamics, integration operator values
- Current step and increment numbers

UEL Variables

Must be Defined	Right-hand-side vector (residual nodal fluxes or forces)	`RHS(MLVARX,*)`
	Jacobian (stiffness) matrix	`AMATRX(NDOFEL,NDOFEL)`
	Solution-dependent state variables	`SVARS(*)`
May be Defined	Energies associated with the element (strain energy, plastic dissipation, kinetic energy, etc.)	`ENERGY(8)`
	Suggested new (reduced) time increment	`PNEWDT`

UEL Conventions

The solution variables (displacement, velocity, etc.) are arranged on a node/degree of freedom basis

The degrees of freedom of the first node are first, followed by the degrees of freedom of the second node, etc.

The flux vector and Jacobian matrix must be ordered in the same way

UEL formulation aspects and usage hints

The displacement, velocities, etc. passed into the UEL are in the **global system**, regardless of whether a local nodal transformation is used at any of the nodes.

The flux vector and Jacobian matrix must also be formulated in the **global system**

UEL Ex: 3D Truss

Local Coordinate

$$[k_e]\{u_e\} = \{f_e\}$$

$$[k_e] = \frac{AE}{L}\begin{bmatrix} 1 & -1 \\ -1 & 1 \end{bmatrix}$$

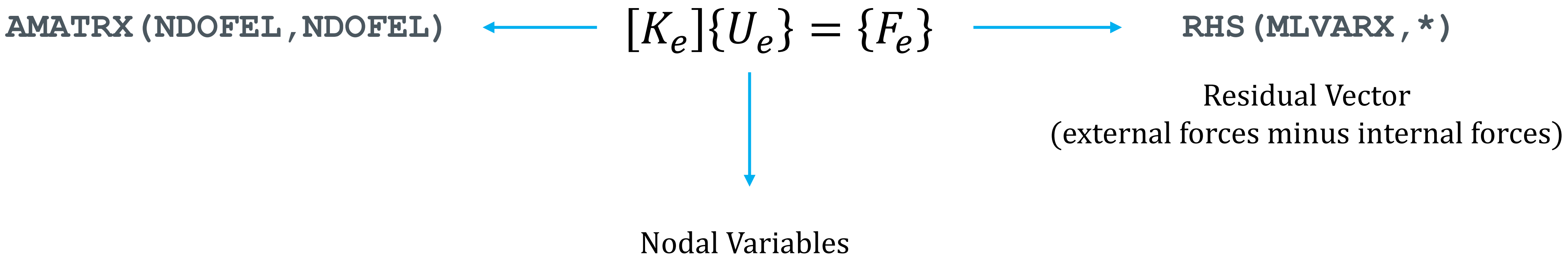
$$[R]\{U_e\} = \{u_e\} \qquad [R]\{F_e\} = \{f_e\} \quad \Rightarrow \quad \left\{\begin{array}{l} [k_e][R]\{U_e\} = [R]\{F_e\} \\ [K_e]\{U_e\} = \{F_e\} \end{array}\right. \quad \Rightarrow \quad [K_e] = [R]^T[k_e][R]$$

$$[R] = \begin{bmatrix} [T] & [0] \\ [0] & [T] \end{bmatrix} = \begin{bmatrix} \frac{x_j - x_i}{L} & \frac{y_j - y_i}{L} & \frac{z_j - z_i}{L} & 0 & 0 & 0 \\ 0 & 0 & 0 & \frac{x_j - x_i}{L} & \frac{y_j - y_i}{L} & \frac{z_j - z_i}{L} \end{bmatrix}$$

$[R]$ × Global Coordinate = Local Coordinate

$[R]^T$ × Local Coordinate = Global Coordinate

UEL Ex: 3D Truss



AMATRX(NDOFEL,NDOFEL)

$$[K_e]\{U_e\} = \{F_e\}$$

RHS(MLVARX,*)

Residual Vector
(external forces minus internal forces)

Nodal Variables

$$[K_e] = [R]^T[k_e][R]$$

$$[R]^T\{f_e\} = \{F_e\}$$

$$[R] = \begin{bmatrix} [T] & [0] \\ [0] & [T] \end{bmatrix} = \begin{bmatrix} \dfrac{x_j - x_i}{L} & \dfrac{y_j - y_i}{L} & \dfrac{z_j - z_i}{L} & 0 & 0 & 0 \\ 0 & 0 & 0 & \dfrac{x_j - x_i}{L} & \dfrac{y_j - y_i}{L} & \dfrac{z_j - z_i}{L} \end{bmatrix}$$

Hints to Write UEL

UEL Variables

Available in UEL

- Coordinates; displacements; incremental displacements; and, for dynamics, velocities and accelerations
- SDVs at the start of the increment
- Total and incremental values of time, temperature, and user-defined field variables
- User element properties
- Load types as well as total and incremental load magnitudes
- Element type and user-defined element number
- Procedure type flag and, for dynamics, integration operator values
- Current step and increment numbers

UEL Variables

Must be Defined	Right-hand-side vector (residual nodal fluxes or forces)	`RHS(MLVARX,*)`
	Jacobian (stiffness) matrix	`AMATRX(NDOFEL,NDOFEL)`
	Solution-dependent state variables	`SVARS(*)`
May be Defined	Energies associated with the element (strain energy, plastic dissipation, kinetic energy, etc.)	`ENERGY(8)`
	Suggested new (reduced) time increment	`PNEWDT`

UEL Conventions

The solution variables (displacement, velocity, etc.) are arranged on a node/degree of freedom basis

The degrees of freedom of the first node are first, followed by the degrees of freedom of the second node, etc.

The flux vector and Jacobian matrix must be ordered in the same way

UEL formulation aspects and usage hints

The displacement, velocities, etc. passed into the UEL are in the **global system**, regardless of whether a local nodal transformation is used at any of the nodes.

The flux vector and Jacobian matrix must also be formulated in the **global system**

UEL formulation aspects and usage hints

The displacement, velocities, etc. passed into the UEL are in the **global system**, regardless of whether a local nodal transformation is used at any of the nodes.

The flux vector and Jacobian matrix must also be formulated in the **global system**

The Jacobian must be formulated as a full matrix, even if it is symmetric

If the UNSYMM parameter is not used, Abaqus will symmetrize the Jacobian defined by the user

For transient heat transfer and dynamic analysis, heat capacity and inertia contributions must be included in the flux vector

UEL formulation aspects and usage hints

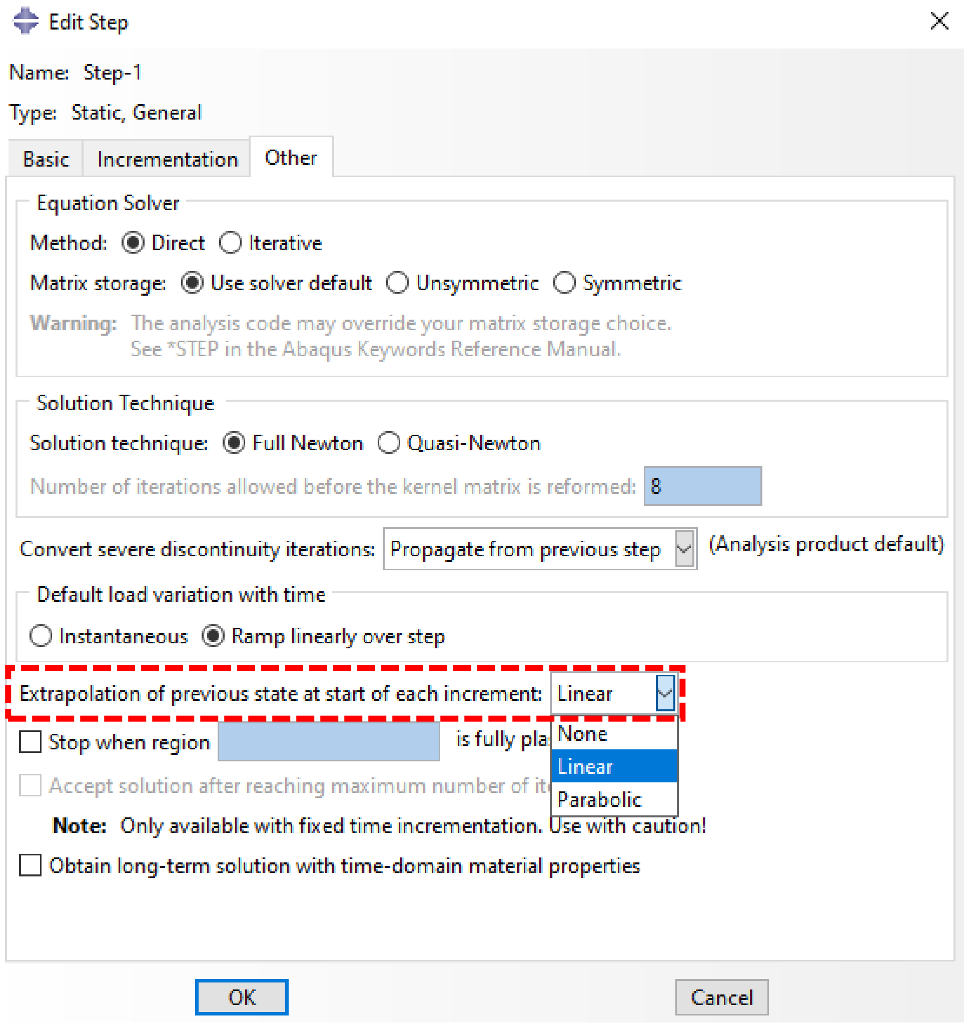
At the start of a new increment, the increment in solution variable(s) is extrapolated from the previous increment.

The flux vector and the Jacobian must be based on these extrapolated values

If extrapolation is not desired, it can be switched off with
 *STEP, EXTRAPOLATION=NO

If the increment in solution variable(s) is too large, the variable PNEWDT can be used to suggest a new time increment.

Abaqus will abandon the current time Ramp linearly over step increment and will attempt the increment again with one that is a factor PNEWDT smaller



Edit Step

Name: Step-1

Type: Static, General

Basic | Incrementation | Other

Equation Solver

Method: ● Direct ○ Iterative

Matrix storage: ● Use solver default ○ Unsymmetric ○ Symmetric

Warning: The analysis code may override your matrix storage choice.
See *STEP in the Abaqus Keywords Reference Manual.

Solution Technique

Solution technique: ● Full Newton ○ Quasi-Newton

Number of iterations allowed before the kernel matrix is reformed: 8

Convert severe discontinuity iterations: Propagate from previous step (Analysis product default)

Default load variation with time

○ Instantaneous ● Ramp linearly over step

Extrapolation of previous state at start of each increment: Linear

None
Linear
Parabolic

□ Stop when region is fully pla

□ Accept solution after reaching maximum number of it

Note: Only available with fixed time incrementation. Use with caution!

□ Obtain long-term solution with time-domain material properties

OK   Cancel

Testing the UEL

Complex UELs may have many potential problem areas. Do not use a large model when trying to debug a UEL

Verify the UEL with a one-element model

- Run tests using general steps in which all solution variables are prescribed to verify the resultant fluxes
- Run tests using linear perturbation steps in which all loads are prescribed to verify the element Jacobian (stiffness)
- Run tests using general steps in which all loads are prescribed to verify the consistency of the Jacobian and the flux vector

Gradually increase the complexity of the test problems. Compare the results with standard Abaqus elements, if possible

UEL Ex: 3D Linear Elastic

Interpolation

$$\left\{\begin{aligned} u &= N_1u_1 + N_2u_2 + \cdots + N_8u_8 \\ v &= N_1v_1 + N_2v_2 + \cdots + N_8v_8 \\ w &= N_1w_1 + N_2w_2 + \cdots + N_8w_8 \end{aligned}\right.$$

⇓

$$\begin{Bmatrix} u \\ v \\ w \end{Bmatrix}_{3\times 1} = \left[\begin{array}{ccc|ccc|c|ccc} N_1 & 0 & 0 & N_2 & 0 & 0 & \dots & N_8 & 0 & 0 \\ 0 & N_1 & 0 & 0 & N_2 & 0 & \dots & 0 & N_8 & 0 \\ 0 & 0 & N_1 & 0 & 0 & N_2 & \dots & 0 & 0 & N_8 \end{array}\right]_{3\times 24} \begin{Bmatrix} u_1 \\ v_1 \\ w_1 \\ u_2 \\ v_2 \\ w_2 \\ . \\ . \\ . \\ u_8 \\ v_8 \\ w_8 \end{Bmatrix}_{24\times 1}$$

⇒

$$\{U\} = [N(\xi,\eta,\zeta)]\{a\}$$

3D Linear Elastic

$$\begin{Bmatrix} \varepsilon_{xx} \\ \varepsilon_{yy} \\ \varepsilon_{zz} \\ \varepsilon_{xy} \\ \varepsilon_{xz} \\ \varepsilon_{yz} \end{Bmatrix}_{6\times 1} = \underbrace{\begin{bmatrix} \frac{\partial}{\partial x} & 0 & 0 \\ 0 & \frac{\partial}{\partial y} & 0 \\ 0 & 0 & \frac{\partial}{\partial z} \\ \frac{\partial}{\partial y} & \frac{\partial}{\partial x} & 0 \\ \frac{\partial}{\partial z} & 0 & \frac{\partial}{\partial x} \\ 0 & \frac{\partial}{\partial z} & \frac{\partial}{\partial y} \end{bmatrix}_{6\times 3}}_{[L]} \begin{Bmatrix} u \\ v \\ w \end{Bmatrix}_{3\times 1}$$

$$\xrightarrow[{[\varepsilon] = [L]\{U\}}]{\{U\} = [N]\{a\}}$$

$$[B] = \underset{6\times 3}{[L]}\,\underset{3\times 24}{[N]}$$

$$\begin{bmatrix} \frac{\partial N_1}{\partial x} & 0 & 0 & \frac{\partial N_2}{\partial x} & 0 & 0 & \cdots & \frac{\partial N_8}{\partial x} & 0 & 0 \\ 0 & \frac{\partial N_1}{\partial y} & 0 & 0 & \frac{\partial N_2}{\partial y} & 0 & \cdots & 0 & \frac{\partial N_8}{\partial y} & 0 \\ 0 & 0 & \frac{\partial N_1}{\partial z} & 0 & 0 & \frac{\partial N_2}{\partial z} & \cdots & 0 & 0 & \frac{\partial N_8}{\partial z} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & 0 & \cdots & \frac{\partial N_8}{\partial y} & \frac{\partial N_8}{\partial x} & 0 \\ \frac{\partial N_1}{\partial z} & 0 & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial z} & 0 & \frac{\partial N_2}{\partial x} & \cdots & \frac{\partial N_8}{\partial z} & 0 & \frac{\partial N_8}{\partial x} \\ 0 & \frac{\partial N_1}{\partial z} & \frac{\partial N_1}{\partial y} & 0 & \frac{\partial N_2}{\partial z} & \frac{\partial N_2}{\partial y} & \cdots & 0 & \frac{\partial N_8}{\partial z} & \frac{\partial N_8}{\partial y} \end{bmatrix}_{6\times 24}$$

Jacobian

$$[B] = [L][N]$$

$$[B] = \begin{bmatrix} \frac{\partial N_i}{\partial x} & 0 & 0 \\ 0 & \frac{\partial N_i}{\partial y} & 0 \\ 0 & 0 & \frac{\partial N_i}{\partial z} \\ \frac{\partial N_i}{\partial y} & \frac{\partial N_i}{\partial x} & 0 \\ \frac{\partial N_i}{\partial z} & 0 & \frac{\partial N_i}{\partial x} \\ 0 & \frac{\partial N_i}{\partial z} & \frac{\partial N_i}{\partial y} \end{bmatrix}$$

$$i = 1\ to\ 8$$

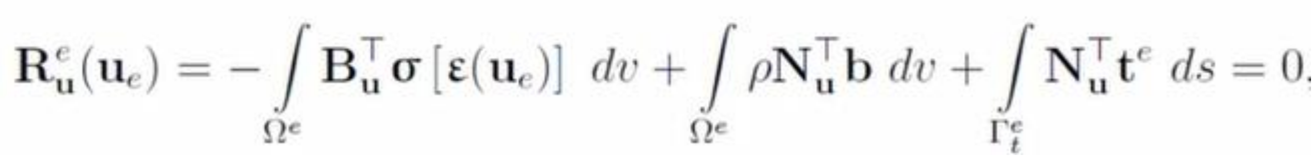
$$\left\{ \frac{\partial N_I}{\partial \xi} \quad \frac{\partial N_I}{\partial \eta} \quad \frac{\partial N_I}{\partial \zeta} \right\} = \left\{ \frac{\partial N_I}{\partial x_1} \quad \frac{\partial N_I}{\partial x_2} \quad \frac{\partial N_I}{\partial x_3} \right\} \begin{bmatrix} \frac{\partial x_1}{\partial \xi} & \frac{\partial x_1}{\partial \eta} & \frac{\partial x_1}{\partial \zeta} \\ \frac{\partial x_2}{\partial \xi} & \frac{\partial x_2}{\partial \eta} & \frac{\partial x_2}{\partial \zeta} \\ \frac{\partial x_3}{\partial \xi} & \frac{\partial x_3}{\partial \eta} & \frac{\partial x_3}{\partial \zeta} \end{bmatrix}$$

$$\frac{\partial N_I}{\partial \boldsymbol{\xi}} = \frac{\partial N_I}{\partial \mathbf{x}} \cdot \mathbf{J}.$$

$$\frac{\partial N_I}{\partial \mathbf{x}} = \frac{\partial N_I}{\partial \boldsymbol{\xi}} \cdot \mathbf{J}^{-1}$$

Introduction to Nonlinear Finite Element Analysis by N. H. Kim

Element Stiffness Matrix



$$\mathbf{R}_{\mathbf{u}}^{e}(\mathbf{u}_e) = -\int_{\Omega^e} \mathbf{B}_{\mathbf{u}}^{\top} \boldsymbol{\sigma}\left[\boldsymbol{\varepsilon}(\mathbf{u}_e)\right] \, dv + \int_{\Omega^e} \rho \mathbf{N}_{\mathbf{u}}^{\top} \mathbf{b} \, dv + \int_{\Gamma_t^e} \mathbf{N}_{\mathbf{u}}^{\top} \mathbf{t}^e \, ds = 0,$$

System of Nonlinear Algebraic Equations

$$\mathbf{P}(\vec{u}) = \vec{f}$$

$$\mathbf{P}(\vec{u}_{i+1}) \approx \mathbf{P}(\vec{u}_i) + \frac{\partial \mathbf{P}(\vec{u}_i)}{\partial \vec{u}_i}\,\Delta\vec{u} = \vec{f}$$

Increment

$$\vec{u}_{i+1} = \vec{u}_i + \Delta\vec{u}$$

$$\mathbf{P}(\vec{u}_{i+1}) \approx \mathbf{P}(\vec{u}_i) + \mathbf{K}_T^i(\vec{u}_i)\,\Delta\vec{u} = \vec{f}$$

$$\mathbf{K}_T^i(\vec{u}_i)\,\Delta\vec{u} = \vec{f} - \mathbf{P}(\vec{u}_i)$$

$$\vec{u}_{i+1} = \vec{u}_i + \Delta\vec{u}$$

Steps to Write Linear UEL

1 The header is usually followed by dimensioning of local arrays. It is good practice to define constants via parameters and to include comments

```
DO I_INPT=1,  NINPT
```

2 Shape Functions and Derivative of shape functions in local coordinates

3 Computing a Jacobian matrix and a determinant of Jacobian matrix

4 Derivative of shape functions in global coordinates

5 Form [B] matrix

6 Computing a stiffness matrix

```
END DO
```

3D Linear Elastic

In reality all solids are three-dimensional. Fortunately, for many practical problems, some simplifying assumptions can be made regarding the stress or strain distributions.

Such as Plane Stress, Plane Strain, and axisymmetric (**symmetry of revolution in both geometry and loading**) Problems

Plane Stress

$$\begin{Bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \tau_{xy} \end{Bmatrix} = \frac{E}{1-\nu^2} \begin{bmatrix} 1 & \nu & 0 \\ \nu & 1 & 0 \\ 0 & 0 & \dfrac{(1-\nu)}{2} \end{bmatrix} \begin{Bmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ \gamma_{xz} \end{Bmatrix}$$

$$\sigma_{zz} = 0 \quad and \quad \varepsilon_{zz} \neq 0$$

Plane Strain

$$\begin{Bmatrix} \sigma_{xx} \\ \sigma_{yy} \\ \tau_{xy} \end{Bmatrix} = \frac{E}{(1+\nu)(1-2\nu)} \begin{bmatrix} 1-\nu & -\nu & 0 \\ -\nu & 1-\nu & 0 \\ 0 & 0 & \dfrac{(1-2\nu)}{2} \end{bmatrix} \begin{Bmatrix} \epsilon_{xx} \\ \epsilon_{yy} \\ \gamma_{xy} \end{Bmatrix}$$

$$\sigma_{zz} \neq 0 \quad and \quad \varepsilon_{zz} = 0$$

$$\begin{pmatrix} \sigma_{11} \\ \sigma_{22} \\ \sigma_{33} \\ \tau_{12} \\ \tau_{13} \\ \tau_{23} \end{pmatrix} = \begin{bmatrix} D_{1111} & D_{1122} & D_{1133} & D_{1112} & D_{1113} & D_{1123} \\ & D_{2222} & D_{2233} & D_{2212} & D_{2213} & D_{2223} \\ & & D_{3333} & D_{3312} & D_{3313} & D_{3323} \\ & \text{symm.} & & D_{1212} & D_{1213} & D_{1223} \\ & & & & D_{1313} & D_{1323} \\ & & & & & D_{2323} \end{bmatrix} \begin{pmatrix} \varepsilon_{11} \\ \varepsilon_{22} \\ \varepsilon_{33} \\ \gamma_{12} \\ \gamma_{13} \\ \gamma_{23} \end{pmatrix}$$

3D Linear Elastic

By substitution

$$\begin{cases} \{\boldsymbol{\varepsilon}\} = [\boldsymbol{L}]\{\boldsymbol{U}\} \\ \{\boldsymbol{U}\} = [\boldsymbol{N}]\{\boldsymbol{a}\} \end{cases}$$

$$\{\boldsymbol{\varepsilon}\} = [\boldsymbol{L}][\boldsymbol{N}]\{\boldsymbol{a}\}= [\boldsymbol{B}]\{\boldsymbol{a}\}$$

$$[B] = \left[\begin{array}{cc|cc|c|cc} \frac{\partial N_1}{\partial x} & 0 & \frac{\partial N_2}{\partial x} & 0 & \dots & \frac{\partial N_n}{\partial x} & 0 \\ 0 & \frac{\partial N_1}{\partial y} & 0 & \frac{\partial N_2}{\partial y} & \dots & 0 & \frac{\partial N_n}{\partial y} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & \dots & \frac{\partial N_n}{\partial y} & \frac{\partial N_n}{\partial x} \end{array}\right]$$

3D Linear Elastic

Variational Approach

$$\int_{V_e} \delta\{\epsilon\}^T\{\sigma\}\,dV = \int_{V_e}\delta\{U\}^T\{b\}\,dV + \int_{\Gamma_e}\delta\{U\}^T\{t\}\,d\Gamma + \sum_i \delta\{U\}^T_{(\{x\}=\{\bar{x}\})}\{P\}_i$$

⇓

$$\{\delta\epsilon\} = \delta([B]\{a\}) = [B]\{\delta a\} \qquad \{\delta U\} = \delta([N]\{a\}) = [N]\{\delta a\} \qquad \{\sigma\} = [D]\{\epsilon\} = [D][B]\{a\}$$

⇓

$$\left[\int_{A_e}[B]^T[D][B]tdA\right]\{a\} = \int_{A_e}[N]^T\{b\}tdA + \int_{L_e}[N]^T\{t\}tdl + \sum_i [N_{(\{x\}=\{\bar{x}\})}]^T\{P\}_i$$

⇓

$$[K_e] = \left[\int_{A_e}[B]^T[D][B]tdA\right] \qquad \{f_e\} = \int_{A_e}[N]^T\{b\}tdA + \int_{L_e}[N]^T\{t\}tdl + \sum_i [N_{(\{x\}=\{\bar{x}\})}]^T\{P\}_i$$

⇓

$$[K_e]\{a\} = f_e$$

3D Linear Elastic

Interpolation

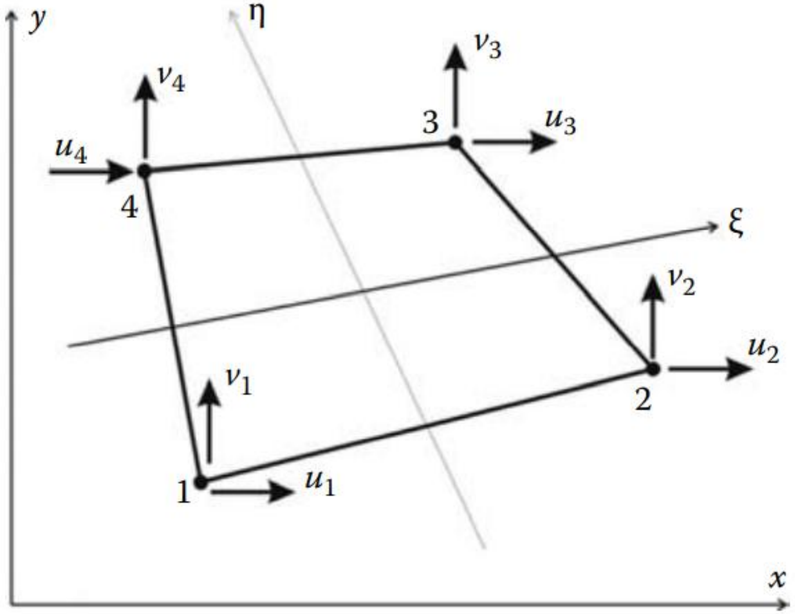
Four node Iso-parametric Element

$$N_1(\xi,\eta) = 0.25(1 - \xi - \eta + \xi\eta)$$

$$N_2(\xi,\eta) = 0.25(1 + \xi - \eta - \xi\eta)$$

$$N_3(\xi,\eta) = 0.25(1 + \xi + \eta + \xi\eta)$$

$$N_4(\xi,\eta) = 0.25(1 - \xi + \eta - \xi\eta)$$



$$u = N_1u_1 + N_2u_2 + N_3u_3 + N_4u_4$$

$$v = N_1v_1 + N_2v_2 + N_3v_3 + N_4v_4$$

3D Linear Elastic

Stiffness Matrix

$$\begin{Bmatrix} u \\ v \end{Bmatrix} = \begin{bmatrix} N_1 & 0 & | & N_2 & 0 & | & \ldots & \ldots & | & N_4 & 0 \\ 0 & N_1 & | & 0 & N_2 & | & \ldots & \ldots & | & 0 & N_4 \end{bmatrix} \begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ \vdots \\ \vdots \\ u_4 \\ v_4 \end{Bmatrix} \Longrightarrow \{U\} = [N]\{a\} \xrightarrow{\{\epsilon\} = [L]\{U\}} \{\epsilon\} = [B]\{a\}$$

$$[L][N] = [B] = \begin{bmatrix} \frac{\partial N_1}{\partial x} & 0 & | & \frac{\partial N_2}{\partial x} & 0 & | & \frac{\partial N_3}{\partial x} & 0 & | & \frac{\partial N_4}{\partial x} & 0 \\ 0 & \frac{\partial N_1}{\partial y} & | & 0 & \frac{\partial N_2}{\partial y} & | & 0 & \frac{\partial N_3}{\partial y} & | & 0 & \frac{\partial N_4}{\partial y} \\ \frac{\partial N_1}{\partial y} & \frac{\partial N_1}{\partial x} & | & \frac{\partial N_2}{\partial y} & \frac{\partial N_2}{\partial x} & | & \frac{\partial N_3}{\partial y} & \frac{\partial N_3}{\partial x} & | & \frac{\partial N_4}{\partial y} & \frac{\partial N_4}{\partial x} \end{bmatrix}$$

3D Linear Elastic

Stiffness Matrix

$$\begin{Bmatrix} u \\ v \end{Bmatrix} = \left[\begin{array}{cc|cc|cc|cc} N_1 & 0 & N_2 & 0 & N_3 & 0 & N_4 & 0 \\ 0 & N_1 & 0 & N_2 & 0 & N_3 & 0 & N_4 \end{array}\right] \begin{Bmatrix} u_1 \\ v_1 \\ u_2 \\ v_2 \\ u_3 \\ v_3 \\ u_4 \\ v_4 \end{Bmatrix} \Longrightarrow \{U\} = [N]\{a\} \Longrightarrow \{\epsilon\} = [B]\{a\}$$

$$\begin{Bmatrix} \dfrac{\partial N_i}{\partial \xi} \\ \dfrac{\partial N_i}{\partial \eta} \end{Bmatrix} = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} \end{bmatrix} \begin{Bmatrix} \dfrac{\partial N_i}{\partial x} \\ \dfrac{\partial N_i}{\partial y} \end{Bmatrix}$$

$$[J] = \begin{bmatrix} \dfrac{\partial x}{\partial \xi} & \dfrac{\partial y}{\partial \xi} \\ \dfrac{\partial x}{\partial \eta} & \dfrac{\partial y}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \displaystyle\sum_{i=1}^{4} \frac{\partial N_i}{\partial \xi} x_i & \displaystyle\sum_{i=1}^{4} \frac{\partial N_i}{\partial \xi} y_i \\ \displaystyle\sum_{i=1}^{4} \frac{\partial N_i}{\partial \eta} x_i & \displaystyle\sum_{i=1}^{4} \frac{\partial N_i}{\partial \eta} y_i \end{bmatrix}$$

$$[J] = \begin{bmatrix} \dfrac{\partial N_1}{\partial \xi} & \dfrac{\partial N_2}{\partial \xi} & \cdots & \dfrac{\partial N_4}{\partial \xi} \\ \dfrac{\partial N_1}{\partial \eta} & \dfrac{\partial N_2}{\partial \eta} & \cdots & \dfrac{\partial N_4}{\partial \eta} \end{bmatrix} \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_4 & y_4 \end{bmatrix}$$

$$x = N_1x_1 + N_2x_2 + N_3x_3 + N_4x_4$$

$$y = N_1y_1 + N_2y_2 + N_3y_3 + N_4y_4$$

$$\begin{Bmatrix} \dfrac{\partial N_i}{\partial x} \\ \dfrac{\partial N_i}{\partial y} \end{Bmatrix} = [J]^{-1} \begin{Bmatrix} \dfrac{\partial N_i}{\partial \xi} \\ \dfrac{\partial N_i}{\partial \eta} \end{Bmatrix}$$

3D Linear Elastic

Stiffness Matrix

$$\begin{Bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \end{Bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} \begin{Bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \end{Bmatrix}$$

$$[J] = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{4} \frac{\partial N_i}{\partial \xi} x_i & \sum_{i=1}^{4} \frac{\partial N_i}{\partial \xi} y_i \\ \sum_{i=1}^{4} \frac{\partial N_i}{\partial \eta} x_i & \sum_{i=1}^{4} \frac{\partial N_i}{\partial \eta} y_i \end{bmatrix}$$

$$[J] = \begin{bmatrix} \frac{\partial N_1}{\partial \xi} & \frac{\partial N_2}{\partial \xi} & \cdots & \frac{\partial N_4}{\partial \xi} \\ \frac{\partial N_1}{\partial \eta} & \frac{\partial N_2}{\partial \eta} & \cdots & \frac{\partial N_4}{\partial \eta} \end{bmatrix} \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ \vdots & \vdots \\ x_4 & y_4 \end{bmatrix}$$

$$x = N_1 x_1 + N_2 x_2 + N_3 x_3 + N_4 x_4$$

$$y = N_1 y_1 + N_2 y_2 + N_3 y_3 + N_4 y_4$$

$$[J] = \frac{1}{4} \begin{bmatrix} -(1-\eta) & (1-\eta) & (1+\eta) & -(1+\eta) \\ -(1-\xi) & -(1+\xi) & (1+\xi) & (1-\xi) \end{bmatrix} \begin{bmatrix} x_1 & y_1 \\ x_2 & y_2 \\ x_3 & y_3 \\ x_4 & y_4 \end{bmatrix}$$

$$\begin{Bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \end{Bmatrix} = \begin{bmatrix} \frac{\partial x}{\partial \xi} & \frac{\partial y}{\partial \xi} \\ \frac{\partial x}{\partial \eta} & \frac{\partial y}{\partial \eta} \end{bmatrix} \begin{Bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \end{Bmatrix}$$

$$\begin{Bmatrix} \frac{\partial N_i}{\partial x} \\ \frac{\partial N_i}{\partial y} \end{Bmatrix} = [J]^{-1} \begin{Bmatrix} \frac{\partial N_i}{\partial \xi} \\ \frac{\partial N_i}{\partial \eta} \end{Bmatrix}$$

3D Linear Elastic

Stiffness Matrix

$$[K_e]\{a\} = f_e$$

$$[K_e] = \left[\int_{A_e} [B]^T [D][B] t\, \underbrace{dA}\right]$$

$$\{f_e\} = \int_{A_e} [N]^T \{b\} t\, dA + \int_{L_e} [N]^T \{t\} t\, dl + \sum_i [N_{(\{x\}=\{\bar{x}\})}]^T \{P\}_i$$

Next Slide

$$[K_e] = t \int_{-1}^{+1} \int_{-1}^{+1} [B(\xi,\eta)]^T [D][B(\xi,\eta)] \overbrace{det[J(\xi,\eta)] d\eta\, d\xi}$$

$$= t \sum_{i=1}^{ngp} \sum_{j=1}^{ngp} W_i W_j [B(\xi_i,\eta_j)]^T [D][B(\xi_i,\eta_j)] det[J(\xi_i,\eta_j)]$$

```
DO iIntPt = 1, nIntPt
wi =
    DO jIntPt = 1, nIntPt
    wj =
        DO kIntPt = 1, nIntPt
        wk =



        END
    END
END
```

Plane Stress Problem: Q4

a. Loop over the Integration points $i = 1$ to nIntPt

b. Retrieve the weight wi as samp(ig, 2)

c. Loop over the Integration points $jg = 1$ to nIntPt

d. Retrieve the weight wj as samp(jg, 2)

e. Loop over the Integration points $jg = 1$ to nIntPt

f. Retrieve the weight wk as samp(jg, 2)

g. Use the function fmlin.m to compute the shape functions, vector fun, and their derivatives, matrix der, in local

h. coordinates, $\xi = \text{samp}(ig, 1)$ and $\eta = \text{samp}(jg, 1)$.

i. Evaluate the Jacobian $jac = der * coord$ v. Evaluate the determinant of the Jacobian as $d = \det(jac)$

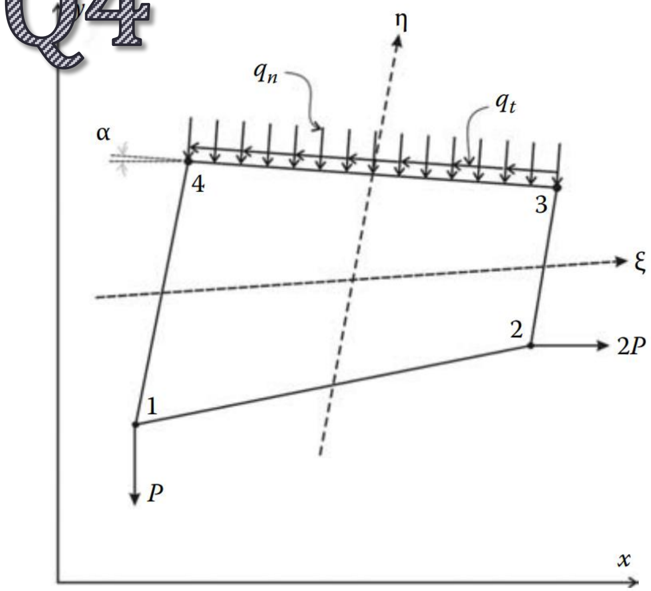
j. Compute the inverse of the Jacobian as $jac1 = inv(jac)$

k. Compute the derivatives of the shape functions with respect to the global coordinates x and y as $deriv = jac1 * der$

l. Use the function formbee.m to form the strain matrix bee ix. Compute the stiffness matrix as
$ke = ke + d * thick * wi * wj * B * D * B$

4. Assemble the stiffness matrix ke into the global matrix kk

Plane Stress Problem: Q4



Body Forces

$$\int_{A_e}[N]^T\{b\}t\,dA = t\sum_{i=1}^{ngp}\sum_{j=1}^{ngp}W_iW_j[N(\xi_i,\eta_j)]^T\begin{Bmatrix}0\\-\rho g\end{Bmatrix}det[J(\xi_i,\eta_j)]$$

Traction Forces

$$q_x = q_t dL\cos\alpha - q_n dL\sin\alpha = q_t dx - q_n dy$$

$$q_y = q_n dL\cos\alpha + q_t dL\sin\alpha = q_n dx + q_t dy$$

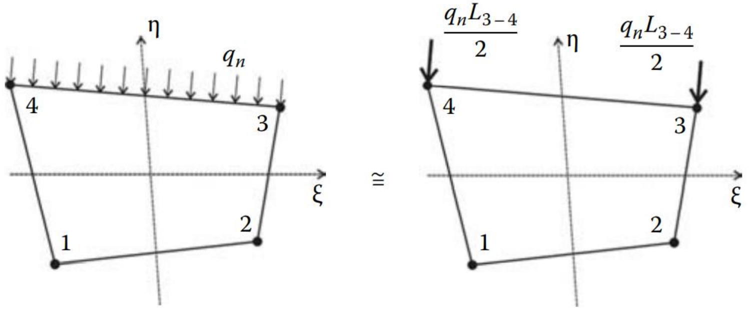
$$q_x = \left(q_t\frac{\partial x}{\partial\xi} - q_n\frac{\partial y}{\partial\xi}\right)d\xi$$

$$q_y = \left(q_n\frac{\partial x}{\partial\xi} + q_t\frac{\partial y}{\partial\xi}\right)d\xi$$

$$\int_{A_e}[N]^T\begin{Bmatrix}q_x\\q_y\end{Bmatrix}dA = t\int_{L_{3-4}}[N(\xi,+1)]^T\begin{Bmatrix}q_x\\q_y\end{Bmatrix}dl$$

$$= t\sum_{i=1}^{ngp}W_i[N(\xi_i,+1)]^T\left[\begin{matrix}\left(q_t\dfrac{\partial x(\xi_i,+1)}{\partial\xi} - q_n\dfrac{\partial y(\xi_i,+1)}{\partial\xi}\right)\\\left(q_n\dfrac{\partial x(\xi_i,+1)}{\partial\xi} + q_t\dfrac{\partial y(\xi_i,+1)}{\partial\xi}\right)\end{matrix}\right]$$

When the nodes of an element are numbered anticlockwise a tangential force, such as q_t, is positive if it acts anticlockwise. A normal force, such as q_n, is positive if it acts toward the interior of the element



In practice, when the loads are uniformly distributed they are replaced by equivalent nodal loads. The preceding development is to be used only if the shape of the loading is complicated.

Concentrated Forces

$$\sum_{k=1}[N]_{x=x_k}\{P_k\} = \begin{bmatrix}1&0\\0&1\\0&0\\0&0\\0&0\\0&0\\0&0\\0&0\end{bmatrix}\begin{Bmatrix}0\\-P\end{Bmatrix} = \begin{bmatrix}0&0\\0&0\\1&0\\0&1\\0&0\\0&0\\0&0\\0&0\end{bmatrix}\begin{Bmatrix}2P\\0\end{Bmatrix} = \begin{Bmatrix}0\\-P\\2P\\0\\0\\0\\0\\0\end{Bmatrix}$$

Plane Stress Problem: Q4

Apply B.C's and Solve (free) Nodal Displacement

$$\begin{bmatrix} [K_{PP}] & \vdots & [K_{PF}] \\ \cdots & \cdots & \cdots \\ [K_{FP}] & \vdots & [K_{FF}] \end{bmatrix} \begin{Bmatrix} \{\delta_P\} \\ \cdots \\ \{\delta_F\} \end{Bmatrix} = \begin{Bmatrix} \{F_P\} \\ \cdots \\ \{F_F\} \end{Bmatrix} \Rightarrow$$

$$[K_{PP}]\{\delta_P\} + [K_{PF}]\{\delta_F\} = \{F_P\}$$

$$[K_{FP}]\{\overset{\to 0}{\cancel{\delta_P}}\} + \underline{[K_{FF}]\{\delta_F\} = \{F_F\}}$$

linsolve(K_FF , F_F)

$$\Rightarrow \{\delta_F\} = [K_{FF}]^{-1}\left\{\{F_F\} - [K_{FP}]\{\delta_P\}\right\}$$

If $\{\delta_p\} = 0$

$$\{\delta_F\} = [K_{FF}]^{-1}\{F_F\}$$

The subscripts P and F refer respectively to the prescribed and free degrees of freedom

Plane Stress Problem: Q4

Calculation of the Element Resultants

SUPPORT REACTIONS

$$[K_{PP}]\{\delta_P\} + [K_{PF}]\{\delta_F\} = \{F_P\} \xrightarrow{\text{If } \{\delta_p\} = 0} \{F_P\} = [K_{PF}]\{\delta_F\}$$

Plane Stress Problem: Q4

Calculation of the Element Resultants

Once the global system of equations is solved, we will compute the stresses at the centroid of the elements. For this we set ngp = 1.

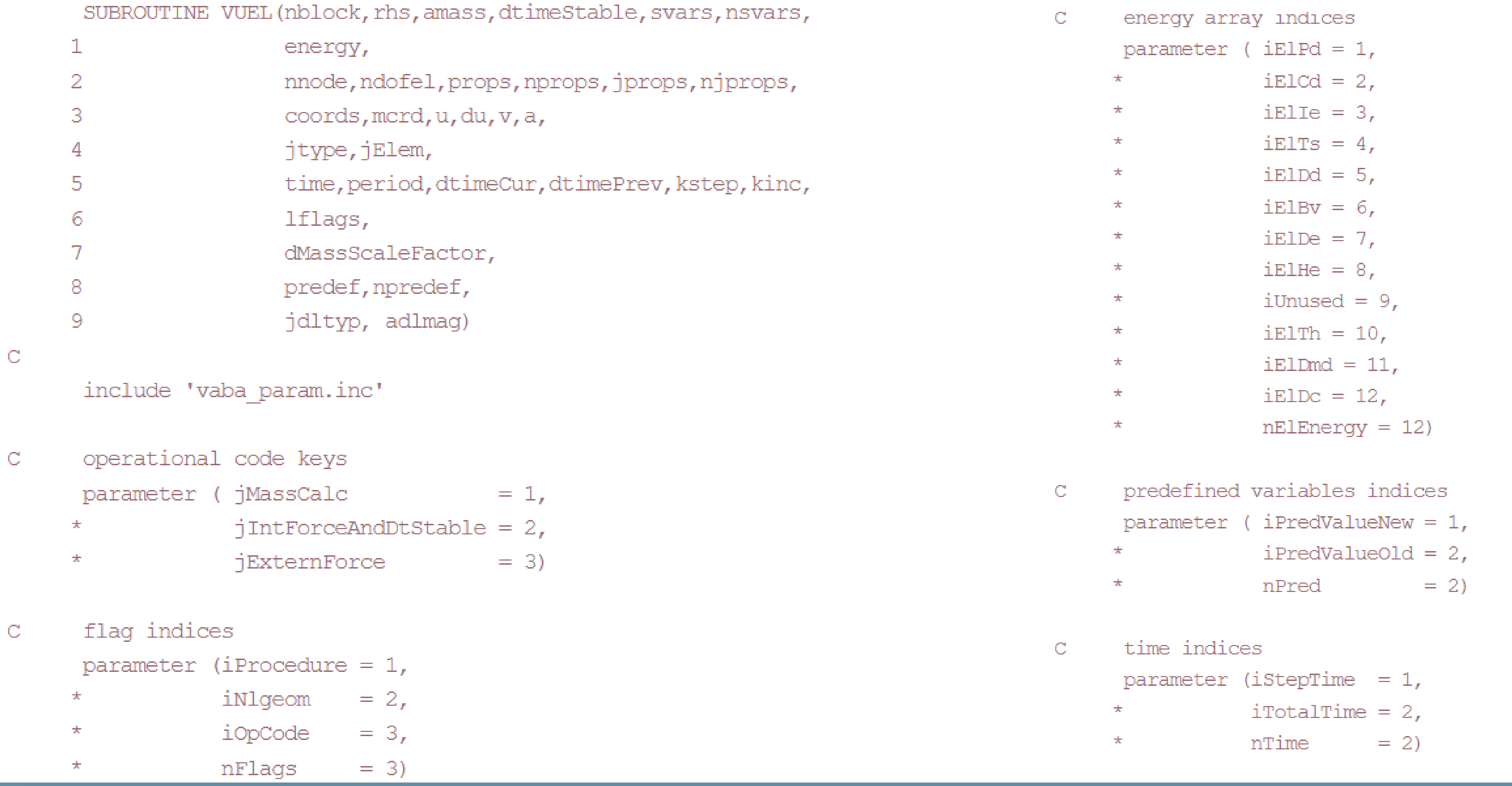
1. For each element
2. Retrieve the coordinates of its nodes coord(nne, 2) and its steering vector g(eldof) using the function elem_Q4.m
3. Retrieve its nodal displacements eld(eldof) from the global vector of displacements delta(n)

a. Loop over the Gauss points ig = 1 to ngp
b. Loop over the Gauss points jg = 1 to ngp
c. Use the function fmlin.m to compute the shape functions, vector fun, and their local derivatives, der, at the local coordinates $\xi = \text{samp}(ig, 1)$ and $\eta = \text{samp}(jg, 1)$
d. Evaluate the Jacobian jac = der ∗ coord
e. Evaluate the determinant of the Jacobian as d = det(jac)
f. Compute the inverse of the Jacobian as jac1 = inv(jac)
g. Compute the derivatives of the shape functions with respect to the global coordinates x and y as deriv = jac1 ∗ der
h. Use the function formbee.m to form the strain matrix bee
i. Compute the strains as eps = bee ∗ eld
j. Compute the stresses as sigma = dee ∗ eps

4. Store the stresses in the matrix SIGMA(nel, 3)

VUEL

Abaqus User Subroutine To Define An (Nonlinear) Element



```
      SUBROUTINE VUEL(nblock,rhs,amass,dtimeStable,svars,nsvars,
     1                energy,
     2                nnode,ndofel,props,nprops,jprops,njprops,
     3                coords,mcrd,u,du,v,a,
     4                jtype,jElem,
     5                time,period,dtimeCur,dtimePrev,kstep,kinc,
     6                lflags,
     7                dMassScaleFactor,
     8                predef,npredef,
     9                jdltyp, adlmag)
C
      include 'vaba_param.inc'

C     operational code keys
      parameter ( jMassCalc            = 1,
     *            jIntForceAndDtStable = 2,
     *            jExternForce         = 3)

C     flag indices
      parameter (iProcedure = 1,
     *           iNlgeom    = 2,
     *           iOpCode    = 3,
     *           nFlags     = 3)
```

```
C     energy array indices
      parameter ( iElPd = 1,
     *            iElCd = 2,
     *            iElIe = 3,
     *            iElTs = 4,
     *            iElDd = 5,
     *            iElBv = 6,
     *            iElDe = 7,
     *            iElHe = 8,
     *            iUnused = 9,
     *            iElTh = 10,
     *            iElDmd = 11,
     *            iElDc = 12,
     *            nElEnergy = 12)

C     predefined variables indices
      parameter ( iPredValueNew = 1,
     *            iPredValueOld = 2,
     *            nPred         = 2)

C     time indices
      parameter (iStepTime  = 1,
     *           iTotalTime = 2,
     *           nTime      = 2)
```

Variables Passed in for Information

DTIME → Time increment

PERIOD → Time period of the current step

NDOFEL → Number of degrees of freedom in the element

MLVARX → Dimensioning parameter used when **several displacement** or **right-hand-side** vectors are used

```
RHS(MLVARX,*), DU(MLVARX,*)
```

NRHS → Number of load vectors

- NRHS=1 in most nonlinear problems
- NRSH=2 for the modified Riks static procedure
- Greater than 1 in some linear analysis procedures and during substructure generation

↓

For example, in the recovery path for the **direct steady-state** procedure, it is 2 to accommodate the **real** and **imaginary** parts of the vectors

Variables Passed in for Information

`NSVARS` --→ User-defined **number of solution-dependent state variables** associated with the element

`NPROPS` --→ User-defined **number of real property** values associated with the element

`NJPROP` --→ User-defined **number of integer property** values associated with the element

`MCRD <= 3` --→ The maximum of

- Maximum number of coordinates required at any node point
- Value of the largest active degree of freedom

`NNODE` --→ User-defined **number of nodes on the element**

Variables Passed in for Information

JTYPE ---→ Integer defining the element type(n)

Abaqus/Standard	Un	$(n \leq 10000)$
Abaqus/Explicit	VUn	$(n \leq 9000)$

KSTEP ---→ Current step number

KINC ---→ Current increment number

JELEM ---→ User-assigned element number

NDLOAD ---→ Identification number of the distributed load or flux currently **active** on this element

MDLOAD ---→ Total number of distributed loads and/or fluxes defined on this element

NPREDF ---→ Number of predefined field variables, including temperature
For user elements Abaqus/Standard uses one value for each field variable per node

Variables Passed in for Information

PROPS(*) → A **floating point** array containing the NPROPS real property values defined for use with this element. NPROPS is the user-specified number of real property values

JPROPS(*) → An **integer** array containing the NJPROP **integer** property values defined for use with this element. NJPROP is the user-specified number of integer property values

COORDS(MCRD, NNODE) → An array containing the **original coordinates** of the nodes of the element COORDS(K1,K2) is the $K1^{th}$ coordinate of the $K2^{th}$ node of the element

JDLTYP(*) → An array containing the integers used to define distributed load types for the element

- Loads of type Un are identified by the integer value n in JDLTYP
- Loads of type UnNU are identified by the negative integer value −n in JDLTYP

JDLTYP(K1,K2) is the identifier of the $K1^{th}$ distributed load in the $K2^{th}$ load case

For general nonlinear steps: K2 =1

Variables Passed in for Information

LFLAGS(*) → An array containing the flags that define the current **solution procedure and requirements** for element calculations.

LFLAGS(1) → Defines The Procedure Type

General Nonlinear Procedures
LFLAGS(4)=0

1, 2	Static
1	Modified Riks Static Analysis (NRHS=2)
11, 12	Direct-Integration Dynamic Analysis
13	Subspace-Based Dynamic Analysis
21	Quasi-Static Analysis

Linear Perturbation Procedures
LFLAGS(4)=1

1, 2	Static
41	Eigenfrequency Extraction Analysis
95	Direct Steady-State Analysis

Variables Passed in for Information

LFLAGS (1)	Procedure	Comments
1, 2	Static	Automatic/fixed time incrementation
11,12	Dynamic	Automatic/fixed time incrementation
21,22	Visco	Quasi-static; explicit/implicit time integration
31	Heat Transfer	Steady-state
32, 33	Heat Transfer	Transient; fixed/automatic time incrementation
41	Frequency extraction	
61	Geostatic	
62, 63	Soils	Steady-state; fixed/automatic time incrementation
64, 65	Soils	Transient; fixed/automatic time incrementation
71	Coupled thermal-stress	Steady-state
72,73	Coupled thermal-stress	Transient; fixed/automatic time incrementation
75	Coupled thermal-electrical	Steady-state
76,77	Coupled thermal-electrical	Transient; fixed/automatic time incrementation

Variables Passed in for Information

`LFLAGS(*)` → An array containing the flags that define the current **solution procedure and requirements** for element calculations.

`LFLAGS(2)` =

- 0 Small-displacement analysis
- 1 Large-displacement analysis (nonlinear geometric effects included in the step)

Variables Passed in for Information

`LFLAGS(3)=`

1 Normal implicit time incrementation procedure. User subroutine UEL must define the residual vector in RHS and the Jacobian matrix in AMATRX.

2 Define the current stiffness matrix (AMATRX = $K^{NM} = -\frac{\partial F^N}{\partial u^M}$ or $-\frac{\partial G^N}{\partial u^M}$) only

3 Define the current damping matrix (AMATRX = $C^{NM} = -\frac{\partial F^N}{\partial \dot{u}^{\mathrm{M}}}$ or $-\frac{\partial G^N}{\partial \dot{u}^{\mathrm{M}}}$) only

4 Define the current mass matrix (AMATRX = $M^{NM} = -\frac{\partial F^N}{\partial \ddot{u}^M}$) only.
Abaqus/Standard always requests an initial mass matrix at the start of the analysis.

5 Define the **current residual or load vector** (RHS $=F^N$) only

6 Define the current **mass matrix** and the **residual vector** for the initial acceleration calculation (or the calculation of accelerations after impact)

100 Define perturbation quantities for output.
Not available for direct steady-state dynamic and mode-based procedures

Variables Passed in for Information

`LFLAGS(4)=`	0	The step is a general step
	1	The step is a linear perturbation step
`LFLAGS(5)=`	0	The current approximations to u^M, etc. were based on **Newton corrections**
	1	The current approximations were found by **extrapolation** from the previous increment
`LFLAGS(7)=`	1	When the damping matrix flag is set, the **viscous damping** matrix is defined
	2	When the damping matrix flag is set, the **structural damping** matrix is defined

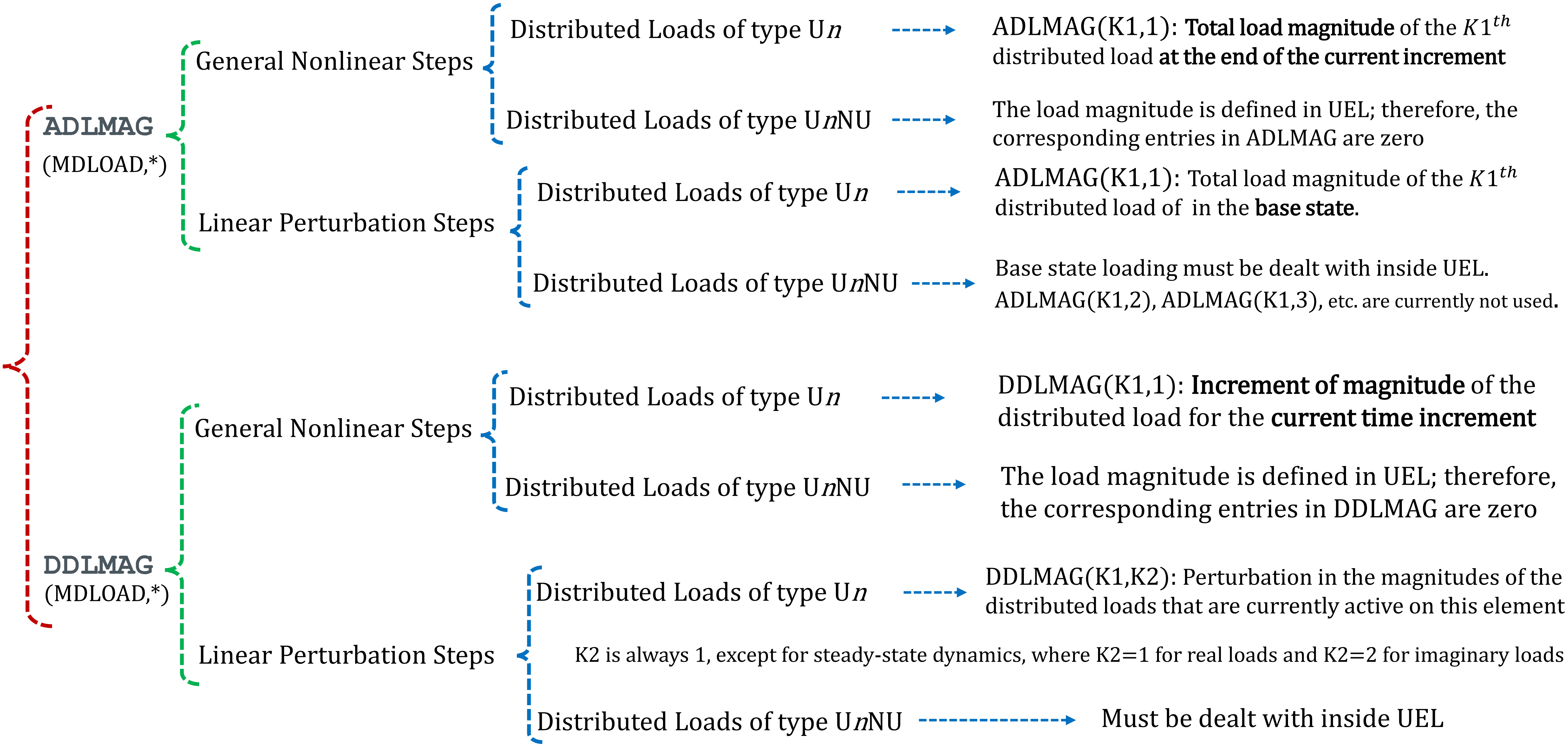
Variables Passed in for Information

```
U, V, A (NDOFEL)
DU(MLVARX,*)
```

Arrays containing the current estimates of the **basic solution variables** (displacements, rotations, temperatures, depending on the degree of freedom) at the nodes of the element at the **end of the current incremen**t. Values are provided as follows:

$U\ (K1\ )$	Total values of the variables. If this is a linear perturbation step, it is the value in the **base state**.
$DU\ (K1, KRHS)$	**Incremental values of the variables for the current increment for right-hand-side KRHS.** For eigenvalue extraction step, this is the eigenvector magnitude for eigenvector KRHS. For steady-state dynamics, KRHS = 1 denotes real components of perturbation displacement and KRHS = 2 denotes imaginary components of perturbation displacement.
$V\ (K1\ )$	Time rate of change of the variables (velocities, rates of rotation). Defined for implicit dynamics only (LFLAGS (1) = 11 or 12).
$A\ (K1\ )$	Accelerations of the variables. Defined for implicit dynamics only (LFLAGS (1) = 11 or 12).

Variables Passed in for Information



- **ADLMAG** (MDLOAD,*)
  - General Nonlinear Steps
    - Distributed Loads of type U*n* → ADLMAG(K1,1): **Total load magnitude** of the $K1^{th}$ distributed load **at the end of the current increment**
    - Distributed Loads of type U*n*NU → The load magnitude is defined in UEL; therefore, the corresponding entries in ADLMAG are zero
  - Linear Perturbation Steps
    - Distributed Loads of type U*n* → ADLMAG(K1,1): Total load magnitude of the $K1^{th}$ distributed load of in the **base state**.
    - Distributed Loads of type U*n*NU → Base state loading must be dealt with inside UEL. ADLMAG(K1,2), ADLMAG(K1,3), etc. are currently not used.
- **DDLMAG** (MDLOAD,*)
  - General Nonlinear Steps
    - Distributed Loads of type U*n* → DDLMAG(K1,1): **Increment of magnitude** of the distributed load for the **current time increment**
    - Distributed Loads of type U*n*NU → The load magnitude is defined in UEL; therefore, the corresponding entries in DDLMAG are zero
  - Linear Perturbation Steps
    - Distributed Loads of type U*n* → DDLMAG(K1,K2): Perturbation in the magnitudes of the distributed loads that are currently active on this element
    - K2 is always 1, except for steady-state dynamics, where K2=1 for real loads and K2=2 for imaginary loads
    - Distributed Loads of type U*n*NU → Must be dealt with inside UEL

Variables Passed in for Information

`PREDEF(2,NPREDF,NNODE)`

An array containing the values of predefined field variables, such as temperature in an uncoupled stress/displacement analysis, at the nodes of the element

Index Of The Array

- First (K1)
  - 1 The value of the field variable at the **end of the increment**
  - 2 The **increment in the field variable**
- Second (K2)
  - 1 The temperature
  - 2, … The predefined field variables
- Third (K3) The local node number on the element

In cases where temperature is not defined, the predefined field variables begin with index 1

PREDEF (K1,1,K3)	Temperature.
PREDEF (K1,2, ,K3)	First predefined field variable.
PREDEF (K1,3, K3)	Second predefined field variable.
Etc.	Any other predefined field variable.
PREDEF (K1,K2, K3)	Total or incremental value of the $K2^{th}$ predefined field variable at the $K3^{th}$ node of the element.
PREDEF (1,K2,K3)	Values of the variables at the end of the current increment.
PREDEF (2,K2,K3)	Incremental values corresponding to the current time increment.

Variables Passed in for Information

PARAMS(*)

An array containing the parameters associated with the **solution procedure**. The entries in this array depend on the solution procedure currently being used when UEL is called, as indicated by the entries in the LFLAGS array.

For implicit dynamics (LFLAGS(1) = 11 or 12) PARAMS contains the **integration operator values**, as:

PARAMS

PARAMS(1) → α

PARAMS(2) → β

PARAMS(3) → γ

TIME(1) Current value of step time or frequency

TIME(2) Current value of total time

Variables to Be Defined

These arrays depend on the value of the **LFLAGS** array

RHS(MLVARX,*) → An array containing the contributions of this element to the right-hand-side vectors of the overall system of equations

AMATRX(NDOFEL,NDOFEL) → An array containing the contribution of this element to the Jacobian (stiffness) or other matrix of the overall system of equations

At time Increment n+1

$$\overset{\text{Residual}}{\mathbf{R}}\left(\mathbf{d}^{\mathrm{n+1}},t^{n+1}\right)=\mathbf{F}_{ext}\left(\mathbf{d}^{\mathrm{n+1}},t^{n+1}\right)-\mathbf{F}_{int}\left(\mathbf{d}^{\mathrm{n+1}},t^{n+1}\right)=0$$

Linearized Model Of The Nonlinear Equations

At time Increment n+1
At Iteration m

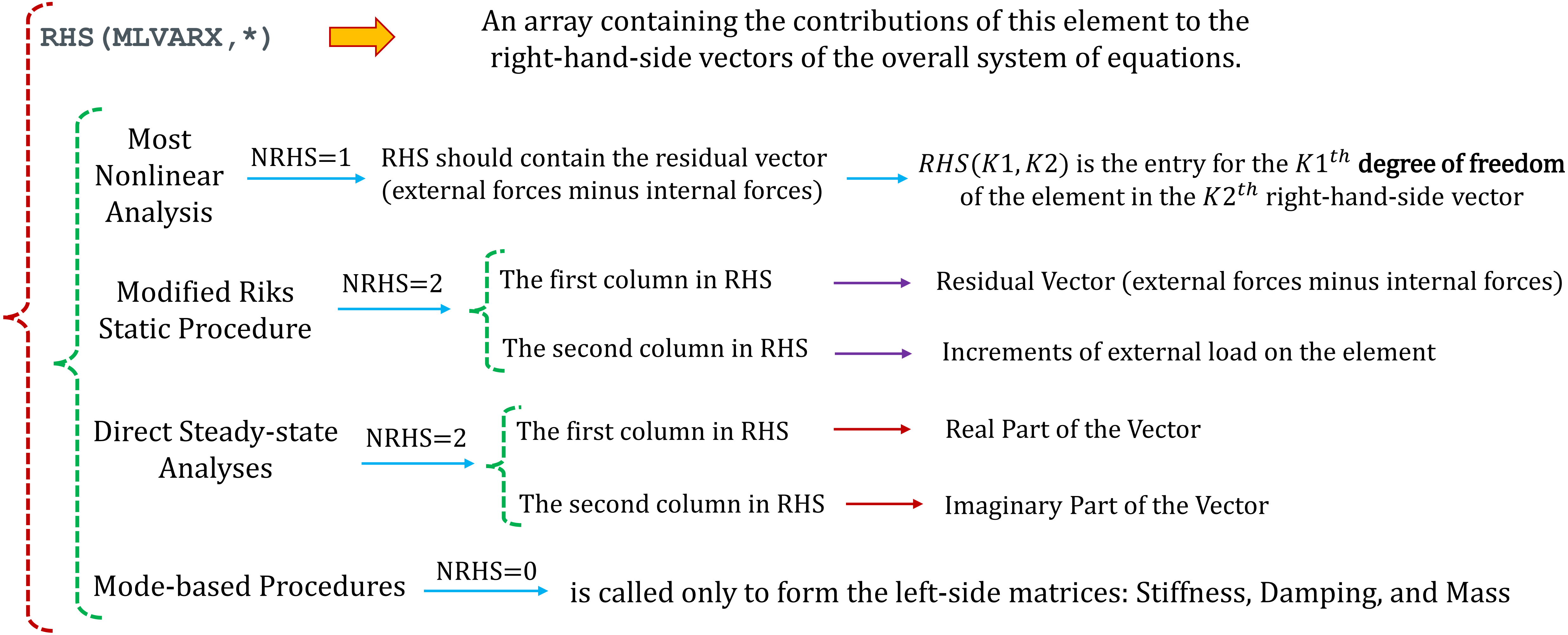
$$\mathbf{R}(\mathbf{d}_{m+1},t^{n+1})=\mathbf{R}(\mathbf{d}_m,t^{n+1})+\underbrace{\frac{\partial\mathbf{R}(\mathbf{d}_m,t^{n+1})}{\partial\mathbf{d}}}_{\text{Jacobian Matrix}}\underbrace{(\mathbf{d}_{\mathrm{m+1}}-\mathbf{d}_{\mathrm{m}})}_{\Delta\mathbf{d}}=0$$

$$\underbrace{\frac{\partial\mathbf{R}(\mathbf{d}_m,t^{n+1})}{\partial\mathbf{d}}}_{\text{NDOFEL× NDOFEL}}\underbrace{\Delta\mathbf{d}}_{\text{DU(MLVARX,*)}}=\mathbf{R}(\mathbf{d}_m,t^{n+1})$$

AMATRX **RHS**

Variables to Be Defined

These arrays depend on the value of the **LFLAGS** array



`RHS(MLVARX,*)` → An array containing the contributions of this element to the right-hand-side vectors of the overall system of equations.

Most Nonlinear Analysis → NRHS=1 → RHS should contain the residual vector (external forces minus internal forces) → $RHS(K1, K2)$ is the entry for the $K1^{th}$ **degree of freedom** of the element in the $K2^{th}$ right-hand-side vector

Modified Riks Static Procedure → NRHS=2

- The first column in RHS → Residual Vector (external forces minus internal forces)
- The second column in RHS → Increments of external load on the element

Direct Steady-state Analyses → NRHS=2

- The first column in RHS → Real Part of the Vector
- The second column in RHS → Imaginary Part of the Vector

Mode-based Procedures → NRHS=0 → is called only to form the left-side matrices: Stiffness, Damping, and Mass

Variables to Be Defined

These arrays depend on the value of the **LFLAGS** array

AMATRX(NDOFEL,NDOFEL) → An array containing the contribution of this element to the Jacobian (stiffness) or other matrix of the overall system of equations

The particular matrix required at any time depends on the entries in the LFLAGS array

All nonzero entries in AMATRX should be defined, even if the matrix is symmetric

- The matrix is unsymmetric → AMATRX
- The matrix is symmetric → AMATRX $= \frac{1}{2}([A] + [A]^T)$

Variables to Be Defined

These arrays depend on the value of the **LFLAGS** array

SVARS(*) → **An array** containing the values of the **solution-dependent state variables** associated with this element

The number of such variables is **NSVARS**

General Nonlinear Steps → This array is passed into UEL containing the values of these variables at the start of the current increment. They should be updated to be the values at the end of the increment, unless the procedure during which UEL is being called does not require such an update.

Linear Perturbation Steps → This array is passed into UEL containing the values of these variables in the **base state**. They should be returned containing perturbation values if you want to output such quantities.

When KINC is equal to zero, the call to UEL is made for zero increment output.
In this case the values returned will be used only for output purposes and are not updated permanently.

Variables to Be Defined

These arrays depend on the value of the **LFLAGS** array

`ENERGY(8)`

General Nonlinear Steps → ENERGY contains the values of the energy quantities associated with the element

⇩

The values in this array when UEL is called are the element energy quantities at the start of the current increment. They should be updated to the values at the end of the **current increment**

Linear Perturbation Steps → ENERGY contains the values of the energy in the **base state**

⇩

They should be returned containing perturbation values if you wish to output such quantities

Mode-based Procedures ⇨ They are not available for updates

Variables to Be Defined

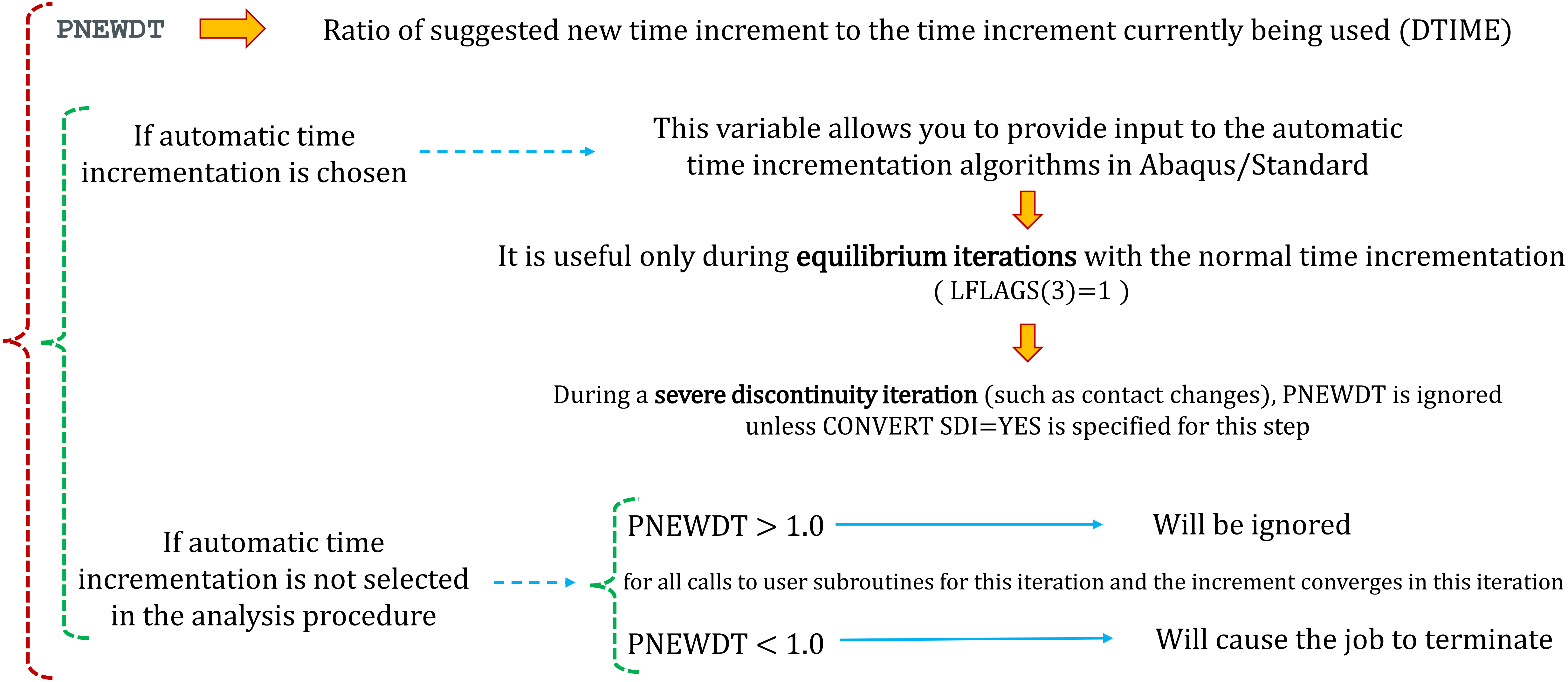
These arrays depend on the value of the **LFLAGS** array

`ENERGY(1)`	→	Kinetic energy
`ENERGY(2)`	→	Elastic strain energy
`ENERGY(3)`	→	Creep dissipation
`ENERGY(4)`	→	Plastic dissipation
`ENERGY(5)`	→	Viscous dissipation
`ENERGY(6)`	→	"Artificial strain energy"
`ENERGY(7)`	→	Electrostatic energy
`ENERGY(8)`	→	Incremental work done by loads applied within the user element

When KINC is equal to zero, the call to UEL is made for zero increment output. In this case the energy values returned will be used only for output purposes and are not updated permanently.

Associated with such effects as artificial stiffness introduced to control hourglassing or other singular modes in the element.

Variables That Can Be Updated



PNEWDT → Ratio of suggested new time increment to the time increment currently being used (DTIME)

If automatic time incrementation is chosen → This variable allows you to provide input to the automatic time incrementation algorithms in Abaqus/Standard

↓

It is useful only during **equilibrium iterations** with the normal time incrementation (LFLAGS(3)=1)

↓

During a **severe discontinuity iteration** (such as contact changes), PNEWDT is ignored unless CONVERT SDI=YES is specified for this step

If automatic time incrementation is not selected in the analysis procedure →

PNEWDT > 1.0 → Will be ignored

for all calls to user subroutines for this iteration and the increment converges in this iteration

PNEWDT < 1.0 → Will cause the job to terminate

Variables That Can Be Updated

If Automatic Time Incrementation Is Chosen:

If PNEWDT is redefined to be less than 1.0

Abaqus/Standard **must** abandon the time increment and attempt it again with a smaller time increment. The suggested new time increment provided to the automatic time integration algorithms is PNEWDT × DTIME, where the PNEWDT used is the minimum value for all calls to user subroutines that allow redefinition of PNEWDT for this iteration

If PNEWDT is given a value that is greater than 1.0

(For all calls to user subroutines for this iteration and the increment converges in this iteration)

Abaqus/Standard **may** increase the time increment. The suggested new time increment provided to the automatic time integration algorithms is PNEWDT × DTIME, where the PNEWDT used is the minimum value for all calls to user subroutines for this iteration.

Continuum Mechanics

Balance Equations: Balance of Mass

$$\frac{Dm}{Dt} = \frac{D(\rho dv)}{Dt} = \iiint_{\Omega} \gamma(x,t)\, dv \quad \Longrightarrow \quad \underbrace{\frac{D}{Dt}\iiint_{\Omega} \rho(x,t)\, dv}_{\iiint_{\Omega}\left[\frac{\partial \rho}{\partial t} + div(\rho v)\right] dv} = \iiint_{\Omega} \gamma(x,t)\, dv$$

Rate Of the Mass Entrance Per Current Volume

Spatial Form:

$$\frac{\partial \rho}{\partial t} + div(\rho v) = \gamma(x,t) \quad \xrightarrow{\gamma(x,t)\,=\,0} \quad \frac{\partial \rho}{\partial t} + div(\rho v) = 0$$

Material Form:

$$\frac{d(J\rho)}{dt} = J\gamma(x,t) \quad \xrightarrow{\gamma(x,t)\,=\,0} \quad \rho_0 = J\rho$$

Continuum Mechanics

Balance Equations: Balance of Linear Momentum

$$\frac{d}{dt}\int_{\Omega}\rho\mathbf{v}\,dv=\oint_{\Gamma}\mathbf{t}\,ds+\int_{\Omega}\mathbf{f}\,dv=\int_{\Omega}\left(\boldsymbol{\nabla}\cdot\boldsymbol{\sigma}\;+\mathbf{f}\right)dv$$

$$\boldsymbol{\nabla}\cdot\boldsymbol{\sigma}\;+\mathbf{f}=\rho\,\frac{d\mathbf{v}}{dt}$$

$$\boldsymbol{\nabla}\cdot\boldsymbol{\sigma}\;+\mathbf{f}=\rho\left(\frac{\partial\mathbf{v}}{\partial t}+\mathbf{v}\cdot\boldsymbol{\nabla}\mathbf{v}\right)$$

$$\boldsymbol{\nabla}_0\cdot\mathbf{P}\;+\mathbf{f}^0=\rho_0\,\frac{\partial\mathbf{V}}{\partial t}$$

$$\boldsymbol{\nabla}_0\cdot\left(\mathbf{S}^{\mathrm{T}}\cdot\mathbf{F}^{\mathrm{T}}\right)+\mathbf{f}^0=\rho_0\,\frac{\partial\mathbf{V}}{\partial t}$$

Continuum Mechanics

Balance Equations: Balance of Angular Momentum

$$\oint_{\Gamma} \mathbf{x} \times \mathbf{t}\, ds + \int_{\Omega} \mathbf{x} \times \mathbf{f}\, dv = \frac{d}{dt} \int_{\Omega} \mathbf{x} \times \rho \mathbf{v}\, dv$$

$$\boldsymbol{\sigma} = \boldsymbol{\sigma}^{\mathrm{T}} \quad \text{or} \quad \sigma_{ij} = \sigma_{ji}$$

Solution Procedures in Total Lagrangian Approach

$$[\mathbf{K}_e(\{\mathbf{u}_e\})]\{\mathbf{u}_e\} = \{\mathbf{F}_e\} \xrightarrow{\text{Iterative procedure}} \{\mathrm{R}\} = [\mathbf{K}_e(\{\mathbf{u}_e\})]\{\mathbf{u}_e\} - \{\mathbf{F}_e\}$$

$$R(u) = R(u^{(r-1)}) + \left(\frac{\partial R}{\partial u}\right)\bigg|_{u^{(r-1)}} \delta u + \frac{1}{2}\left(\frac{\partial^2 R}{\partial u^2}\right)\bigg|_{u^{(r-1)}} (\delta u)^2 + \cdots = 0$$

Where $\delta u^{(r)} = u^{(r)} - u^{(r-1)}$

$$\delta u^{(r)} = -\left(K_T(u^{(r-1)})\right)^{-1} R(u^{(r-1)}) = \left(K_T(u^{(r-1)})\right)^{-1}\left(F - K(u^{(r-1)})\, u^{(r-1)}\right)$$

Where $K_T = \left.\dfrac{\partial R}{\partial u}\right|_{u^{(r-1)}}$

Abaqus Consistent Jacobian

$$\mathbf{K}_{\text{int}} = \iiint\limits_{\Omega_0} \frac{\partial(J\boldsymbol{\sigma} : \delta\mathbf{D})}{\partial\mathbf{D}} dV$$

$$\mathbf{K}_{ijkl} = \iiint\limits_{\Omega_0} \frac{\partial\left(J\sigma_{ij}\,\delta D_{ij}\right)}{\partial D_{kl}} dV = \iiint\limits_{\Omega_0} \left[\frac{\partial J}{\partial D_{kl}}\sigma_{ij}\,\delta D_{ij} + \frac{\partial\sigma_{ij}}{\partial D_{kl}}J\,\delta D_{ij} + \frac{\partial\left(\delta D_{ij}\right)}{\partial D_{kl}}J\sigma_{ij}\right] dV$$

$$\mathbf{K}_{ijkl} = \iiint\limits_{\Omega_0} \left[\frac{\partial J}{\partial D_{kl}}\sigma_{ij}\,\delta D_{ij} + \frac{\partial\sigma_{ij}}{\partial D_{kl}}J\,\delta D_{ij}\right] dV$$

Procedures And Basic Equations

$$\int_V \boldsymbol{\sigma} : \delta \mathbf{D}\, dV = \int_S \mathbf{t}^T \cdot \delta \mathbf{v}\, dS + \int_V \mathbf{f}^T \cdot \delta \mathbf{v}\, dV.$$

$$\int_{V^0} \boldsymbol{\tau}^c : \delta \boldsymbol{\varepsilon}\, dV^0 = \int_S \mathbf{t}^T \cdot \delta \mathbf{v}\, dS + \int_V \mathbf{f}^T \cdot \delta \mathbf{v}\, dV$$

$$\mathbf{u} = \mathbf{N}_N u^N, \quad \delta \mathbf{v} = \mathbf{N}_N \delta v^N \qquad \delta \boldsymbol{\varepsilon} = \boldsymbol{\beta}_N \delta v^N$$

$$\delta v^N \int_{V^0} \boldsymbol{\beta}_N : \boldsymbol{\tau}^c\, dV^0 = \delta v^N \left[\int_S \mathbf{N}_N^T \cdot \mathbf{t}\, dS + \int_V \mathbf{N}_N^T \cdot \mathbf{f}\, dV \right]$$

$$\int_{V^0} \boldsymbol{\beta}_N : \boldsymbol{\tau}^c\, dV^0 = \int_S \mathbf{N}_N^T \cdot \mathbf{t}\, dS + \int_V \mathbf{N}_N^T \cdot \mathbf{f}\, dV.$$

$$F^N \left(u^M \right) = 0$$

$$M^{NM} \ddot{u}^M + F^N \left(u^M \right) = 0.$$