UEL

Warning: This feature is intended for advanced users only. Its use in all but the simplest test examples will require considerable coding by the user/developer. User-defined elements should be read before proceeding.

User subroutine UEL:

- will be called for each element that is of a general user-defined element type (that is, not defined by a linear stiffness, damping, or mass matrix read either directly or from results file data) each time element calculations are required; and
- (or subroutines called by user subroutine UEL) must perform, depending on the analysis type, all or most of the calculations for the element, appropriate to the current activity in the analysis.

The following topics are discussed:

Products: Abaqus/Standard

Wave kinematic data

For Abaqus/Aqua applications four utility routines—GETWAVE, GETWAVEVEL, GETWINDVEL, and GETCURRVEL—are provided to access the fluid kinematic data. These routines are used from within user subroutine UEL and are discussed in detail in Obtaining wave kinematic data in an Abaqus/Aqua analysis.

User subroutine interface

```
      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 to be defined

These arrays depend on the value of the LFLAGS array.

RHS

An array containing the contributions of this element to the right-hand-side vectors of the overall system of equations. For most nonlinear analysis procedures, NRHS=1 and RHS should contain the residual vector (external forces minus internal forces). The exception is the modified Riks static procedure (Static stress analysis), for which NRHS=2 and the first column in RHS should contain the residual vector and the second column should contain the increments of external load on the element. RHS(K1,K2) is the entry for the K1th degree of freedom of the element in the K2 the right-hand-side vector. For direct steady-state analyses, set NRHS=2 in the recovery path for the reaction force to define real and imaginary parts of the vectors. RHS(K1,K2) is the entry for the K1th degree of freedom, K2=1 is the entry for the real part, and K2=2 is the entry for the imaginary part of the vector. For mode-based procedures, this user subroutine is called only to form the left-side matrices: stiffness, damping, and mass. Right-side vectors, as well as the reaction force vector, are calculated outside the user subroutine automatically. The reaction force calculation takes the inertia force and the forces due to the damping into account.

AMATRX

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 (see below).

All nonzero entries in AMATRX should be defined, even if the matrix is symmetric. If you do not specify that the matrix is unsymmetric when you define the user element, Abaqus/Standard will use the symmetric matrix defined by $\frac{1}{2}\left([A]+[A]^T\right)$, where $[A]$ is the matrix defined as AMATRX in this subroutine. If you specify that the matrix is unsymmetric when you define the user element, Abaqus/Standard will use AMATRX directly.

SVARS

An array containing the values of the solution-dependent state variables associated with this element. The number of such variables is NSVARS (see below). You define the meaning of these variables.

For 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. This depends on the entries in the LFLAGS array (see below). For 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 (see About Output). In this case the values returned will be used only for output purposes and are not updated permanently.

ENERGY

For general nonlinear steps array 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. For linear perturbation steps the array is passed into UEL containing the energy in the base state. They should be returned containing perturbation values if you wish to output such quantities. They are not available for updates for mode-based procedures. The entries in the array are as follows:

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” associated with such effects as artificial stiffness introduced to control hourglassing or other singular modes in the element.
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 (see About Output). In this case the energy values returned will be used only for output purposes and are not updated permanently.

Variables that can be updated

PNEWDT

Ratio of suggested new time increment to the time increment currently being used (DTIME, see below). This variable allows you to provide input to the automatic time incrementation algorithms in Abaqus/Standard (if automatic time incrementation is chosen). It is useful only during equilibrium iterations with the normal time incrementation, as indicated by LFLAGS(3)=1. During a severe discontinuity iteration (such as contact changes), PNEWDT is ignored unless CONVERT SDI=YES is specified for this step. The usage of PNEWDT is discussed below.

PNEWDT is set to a large value before each call to UEL.

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.

If automatic time incrementation is not selected in the analysis procedure, values of PNEWDT that are greater than 1.0 will be ignored and values of PNEWDT that are less than 1.0 will cause the job to terminate.

Variables passed in for information

Arrays:

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. See Defining the element properties.

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. See Defining the element properties.

COORDS

An array containing the original coordinates of the nodes of the element. COORDS(K1,K2) is the K1th coordinate of the K2th node of the element.

U, DU, V, A

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 increment. 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. If this is an 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).

JDLTYP

An array containing the integers used to define distributed load types for the element. Loads of type U*n* are identified by the integer value *n* in JDLTYP; loads of type U*n*NU are identified by the negative integer value $-n$ in JDLTYP. JDLTYP(K1,K2) is the identifier of the K1th distributed load in the K2th load case. For general nonlinear steps K2 is always 1.

ADLMAG

For general nonlinear steps ADLMAG(K1,1) is the total load magnitude of the K1th distributed load at the end of the current increment for distributed loads of type U*n*. For distributed loads of type U*n*NU, the load magnitude is defined in UEL; therefore, the corresponding entries in ADLMAG are zero. For linear perturbation steps ADLMAG(K1,1) contains the total load magnitude of the K1th distributed load of type U*n* applied in the base state. Base state loading of type U*n*NU must be dealt with inside UEL. ADLMAG(K1,2), ADLMAG(K1,3), etc. are currently not used.

DDLMAG

For general nonlinear steps DDLMAG contains the increments in the magnitudes of the distributed loads that are currently active on this element for distributed loads of type U*n*. DDLMAG(K1,1) is the increment of magnitude of the load for the current time increment. The increment of load magnitude is required to compute the external work contribution. For distributed loads of type U*n*NU, the load magnitude is defined in UEL; therefore, the corresponding entries in DDLMAG are zero. For linear perturbation steps DDLMAG(K1,K2) contains the perturbation in the magnitudes of the distributed loads that are currently active on this element for distributed loads of type U*n*. K1 denotes the K1th perturbation load active on the element. K2 is always 1, except for steady-state dynamics, where K2=1 for real loads and K2=2 for imaginary loads. Perturbation loads of type U*n*NU must be dealt with inside UEL.

PREDEF

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

The first index of the array, K1, is either 1 or 2, with 1 indicating the value of the field variable at the end of the increment and 2 indicating the increment in the field variable. The second index, K2, indicates the variable: the temperature corresponds to index 1, and the predefined field variables correspond to indices 2 and above. In cases where temperature is not defined, the predefined field variables begin with index 1. The third index, K3, indicates the local node number on the element.

`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 K2th predefined field variable at the K3th 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.

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 (see below).

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

`PARAMS(1)`	α
`PARAMS(2)`	β
`PARAMS(3)`	γ

LFLAGS

An array containing the flags that define the current solution procedure and requirements for element calculations. Detailed requirements for the various Abaqus/Standard procedures are defined earlier in this section.

`LFLAGS(1)`	Defines the procedure type. See Results file for the key used for each procedure.
`LFLAGS(2)=0`	Small-displacement analysis.
`LFLAGS(2)=1`	Large-displacement analysis (nonlinear geometric effects included in the step; see General and perturbation procedures).
`LFLAGS(3)=1`	Normal implicit time incrementation procedure. User subroutine UEL must define the residual vector in RHS and the Jacobian matrix in AMATRX.
`LFLAGS(3)=2`	Define the current stiffness matrix (AMATRX$= K^{NM} = -\partial F^N/\partial u^M \text{ or } -\partial G^N/\partial u^M$) only.
`LFLAGS(3)=3`	Define the current damping matrix (AMATRX$= C^{NM} = -\partial F^N/\partial \dot{u}^M \text{ or } -\partial G^N/\partial \dot{u}^M$) only. To declare the type of damping matrix as viscous or structural, use LFLAGS(7) as well (see below).
`LFLAGS(3)=4`	Define the current mass matrix (AMATRX$= M^{NM} = -\partial F^N/\partial \ddot{u}^M$) only. Abaqus/Standard always requests an initial mass matrix at the start of the analysis.
`LFLAGS(3)=5`	Define the current residual or load vector (RHS $= F^N$) only.
`LFLAGS(3)=6`	Define the current mass matrix and the residual vector for the initial acceleration calculation (or the calculation of accelerations after impact).
`LFLAGS(3)=100`	Define perturbation quantities for output. Not available for direct steady-state dynamic and mode-based procedures.
`LFLAGS(4)=0`	The step is a general step.
`LFLAGS(4)=1`	The step is a linear perturbation step.
`LFLAGS(5)=0`	The current approximations to u^M, etc. were based on Newton corrections.
`LFLAGS(5)=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.

`LFLAGS(7)=2` When the damping matrix flag is set, the structural damping matrix is defined.

`TIME(1)`

Current value of step time or frequency.

`TIME(2)`

Current value of total time.

Scalar parameters:

`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.

`NRHS`

Number of load vectors. `NRHS` is 1 in most nonlinear problems: it is 2 for the modified Riks static procedure (Static stress analysis), and it is 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.

`NSVARS`

User-defined number of solution-dependent state variables associated with the element (Defining the number of solution-dependent variables that must be stored within the element).

`NPROPS`

User-defined number of real property values associated with the element (Defining the element properties).

`NJPROP`

User-defined number of integer property values associated with the element (Defining the element properties).

`MCRD`

`MCRD` is defined as the maximum of the user-defined maximum number of coordinates required at any node point (Defining the maximum number of coordinates needed at any nodal point) and the value of the largest active degree of freedom of the user element that is less than or equal to 3. For example, if you specify that the maximum number of coordinates is 1 and the active degrees of freedom of the user element are 2, 3, and 6, `MCRD` will be 3. If you specify that the maximum number of coordinates is 2 and the active degrees of freedom of the user element are 11 and 12, `MCRD` will be 2.

`NNODE`

User-defined number of nodes on the element (Defining the number of nodes associated with the element).

`JTYPE`

Integer defining the element type. This is the user-defined integer value *n* in element type U*n* (Assigning an element type key to a user-defined element).

`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.

`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.

Usage with general nonlinear procedures

The values of u^N (and, in direct-integration dynamic steps, $\dot{u}^N$ and $\ddot{u}^N$) enter user subroutine UEL as their latest approximations at the end of the time increment; that is, at time $t + \Delta t$.

The values of H^α enter the subroutine as their values at the beginning of the time increment; that is, at time *t*. It is your responsibility to define suitable time integration schemes to update H^α. To ensure accurate, stable integration of internal state variables, you can control the time incrementation via `PNEWDT`.

The values of p^β enter the subroutine as the values of the total load magnitude for the βth distributed load at the end of the increment. Increments in the load magnitudes are also available.

In the following descriptions of the user element's requirements, it will be assumed that `LFLAGS(3)=1` unless otherwise stated.

Static analysis (`LFLAGS(1)=1,2`)

- $F^N = F^N\left(u^M, H^\alpha, p^\beta, t\right)$, where the residual force is the external force minus the internal force.
- Automatic convergence checks are applied to the force residuals corresponding to degrees of freedom 1–7.
- You must define `AMATRX`$= K^{NM} = -\partial F^N / \partial u^M$ and `RHS` $= F^N$ and update the state variables, H^α.

Modified Riks static analysis (`LFLAGS(1)=1`) and (`NRHS=2`)

- $F^N = F^N\left(u^M, H^\alpha, p^\beta\right)$, where $p^\beta = p_0^\beta + \lambda q^\beta$, p_0^β and q^β are fixed load parameters, and λ is the Riks (scalar) load parameter.
- Automatic convergence checks are applied to the force residuals corresponding to degrees of freedom 1–7.
- You must define `AMATRX`$= K^{NM} = -\partial F^N / \partial u^M$, `RHS(1)` $= F^N$, and `RHS(2)` $= \Delta\lambda\left(\partial F^N / \partial\lambda\right)$ and update the state variables, H^α. `RHS(2)` is the incremental load vector.

Direct-integration dynamic analysis (`LFLAGS(1)=11, 12`)

- Automatic convergence checks are applied to the force residuals corresponding to degrees of freedom 1–7.
- `LFLAGS(3)=1`: Normal time increment. Either the Hilber-Hughes-Taylor or the backward Euler time integration scheme will be used. With α set to zero for the backward Euler, both schemes imply

  $$F^N = -M^{NM}\ddot{u}_{\ t+\Delta t} + (1+\alpha)\ G^N{}_{\ t+\Delta t} - \alpha\, G_t^N,$$

  where $M^{NM} = M^{NM}\left(u^M, \dot{u}^M, H^\alpha, p^\beta, t, \ldots\right)$ and $G^N = G^N\left(u^M, \dot{u}^M, H^\alpha, p^\beta, t, \ldots\right)$; that is, the highest time derivative of u^M in M^{NM} and G^N is $\dot{u}^M$, so that

  $$-\frac{\partial F^N}{\partial \ddot{u}^M{}_{t+\Delta t}} \quad = \quad M^{NM}.$$

  Therefore, you must store G_t^N as an internal state vector. If half-increment residual calculations are required, you must also store $G_{t^-}^N$ as an internal state vector, where t^- indicates the time at the beginning of the previous increment. For $\alpha = 0$, $F^N = -M^{NM}\ddot{u}_{\ t+\Delta t} + G^N{}_{t+\Delta t}$ and G_t^N is not needed. You must define `AMATRX` $= M^{NM}\left(d\ddot{u}/du\right) + (1+\alpha)\, C^{NM}\left(d\dot{u}/du\right) + (1+\alpha)\, K^{NM}$, where $C^{NM} = -\partial G^N{}_{t+\Delta t}/\partial\dot{u}^M$ and $K^{NM} = -\partial G^N{}_{t+\Delta t}/\partial u^M$. `RHS` $= F^N$ must also be defined and the state variables, H^α, updated. Although the value of α given in the dynamic step definition is passed into UEL, the value of α can vary from element to element. For example, α can be set to zero for some elements in the model where numerical dissipation is not desired.
- `LFLAGS(3)=5`: Half-increment residual ($F_{1/2}^N$) calculation. Abaqus/Standard will adjust the time increment so that $\max\left|F_{1/2}^N\right| < tolerance$ (where $tolerance$ is specified in the dynamic step definition). The half-increment residual is defined as

$$F^N_{1/2} = -M^{NM}\ddot{u}_{t+\Delta t/2} + (1+\alpha)\,G^N{}_{t+\Delta t/2} - \frac{\alpha}{2}\left(G^N_t + G^N_{t^-}\right),$$

where t^- indicates the time at the beginning of the previous increment (α is a parameter of the Hilber-Hughes-Taylor time integration operator and will be set to zero if the backward Euler time integration operator is used). You must define `RHS` $= F^N_{1/2}$. To evaluate M^{NM} and $G^N{}_{t+\Delta t/2}$, you must calculate $H^\alpha{}_{t+\Delta t/2}$. These half-increment values will not be saved. `DTIME` will still contain Δt (not $\Delta t/2$). The values contained in `U`, `V`, `A`, and `DU` are half-increment values.

- `LFLAGS(3)=4`: Velocity jump calculation. Abaqus/Standard solves $-M^{NM}\,\Delta\dot{u}^M = 0$ for $\Delta\dot{u}^M$, so you must define `AMATRX` $= M^{NM}$.
- `LFLAGS(3)=6`: Initial acceleration calculation. Abaqus/Standard solves $-M^{NM}\ddot{u}^M + G^N = 0$ for $\ddot{u}^M$, so you must define `AMATRX` $= M^{NM}$ and `RHS` $= G^N$.

Subspace-based dynamic analysis (`LFLAGS(1)=13`)

- The requirements are identical to those of static analysis, except that the Jacobian (stiffness), `AMATRX`, is not required. No convergence checks are performed in this case.

Quasi-static analysis (`LFLAGS(1)=21`)

- The requirements are identical to those of static analysis.

Steady-state heat transfer analysis (`LFLAGS(1)=31`)

- The requirements are identical to those of static analysis, except that the automatic convergence checks are applied to the heat flux residuals corresponding to degrees of freedom 11, 12, …

Transient heat transfer analysis ($\Delta\theta_{max}$) (`LFLAGS(1)=32, 33`)

- Automatic convergence checks are applied to the heat flux residuals corresponding to degrees of freedom 11, 12, …
- The backward difference scheme is always used for time integration; that is, Abaqus/Standard assumes that $\dot{u}_{t+\Delta t} = \Delta u/\Delta t$, where $\Delta u = u_{t+\Delta t} - u_t$ and so $d\dot{u}/du = 1/\Delta t$ always. For degrees of freedom 11, 12, …, $\max|\Delta u|$ will be compared against the user-prescribed maximum allowable nodal temperature change in an increment, $\Delta\theta_{max}$, for controlling the time integration accuracy.
- You need to define `AMATRX` $= K^{NM} + (1/\Delta t)\ C^{NM}$, where C^{NM} is the heat capacity matrix and `RHS` $= F^N$, and must update the state variables, H^α.

Geostatic analysis (`LFLAGS(1)=61`)

- Identical to static analysis, except that the automatic convergence checks are applied to the residuals corresponding to degrees of freedom 1–8.

Steady-state coupled pore fluid diffusion/stress analysis (`LFLAGS(1)=62, 63`)

- Identical to static analysis, except that the automatic convergence checks are applied to the residuals corresponding to degrees of freedom 1–8.

Transient coupled pore fluid diffusion/stress (consolidation) analysis (Δu_w^{max}) (`LFLAGS(1)=64, 65`)

- Automatic convergence checks are applied to the residuals corresponding to degrees of freedom 1–8.
- The backward difference scheme is used for time integration; that is, $\dot{u}^M_{t+\Delta t} = \Delta u^M/\Delta t$, where $\Delta u^M = u^M_{t+\Delta t} - u^M_t$.
- For degree of freedom 8, $\max\left|\Delta u^M\right|$ will be compared against the user-prescribed maximum wetting liquid pore pressure change, Δu_w^{max}, for automatic control of the time integration accuracy.
- You must define `AMATRX` $= K^{NM} + (1/\Delta t)\ C^{NM}$, where C^{NM} is the pore fluid capacity matrix and `RHS` $= F^N$, and must update the state variables, H^α.

Steady-state fully coupled thermal-stress analysis (`LFLAGS(1)=71`)

- Identical to static analysis, except that the automatic convergence checks are applied to the residuals corresponding to degrees of freedom 1–7 and 11, 12, …

Transient fully coupled thermal-stress analysis ($\Delta\theta_{max}$) (LFLAGS(1)=72,73)

- Automatic convergence checks are applied to the residuals corresponding to degrees of freedom 1–7 and 11, 12, …
- The backward difference scheme is used for time integration; that is, $\dot{u}^M_{t+\Delta t} = \Delta u^M / \Delta t$, where $\Delta u^M = u^M_{t+\Delta t} - u^M_t$.
- For degrees of freedom 11, 12, …, $\max\left|\Delta u^M\right|$ will be compared against the user-prescribed maximum allowable nodal temperature change in an increment, $\Delta\theta_{max}$, for automatic control of the time integration accuracy.
- You must define AMATRX $= K^{NM} + (1/\Delta t)\ C^{NM}$, where C^{NM} is the heat capacity matrix and RHS $= F^N$, and must update the state variables, H^α.

Steady-state coupled thermal-electrical analysis (LFLAGS(1)=75)

- The requirements are identical to those of static analysis, except that the automatic convergence checks are applied to the current density residuals corresponding to degree of freedom 9, in addition to the heat flux residuals.

Transient coupled thermal-electrical analysis ($\Delta\theta_{max}$) (LFLAGS(1)=76, 77)

- Automatic convergence checks are applied to the current density residuals corresponding to degree of freedom 9 and to the heat flux residuals corresponding to degree of freedom 11.
- The backward difference scheme is always used for time integration; that is, Abaqus/Standard assumes that $\dot{u}_{t+\Delta t} = \Delta u / \Delta t$, where $\Delta u = u_{t+\Delta t} - u_t$. Therefore, $d\dot{u}/du = 1/\Delta t$ always. For degree of freedom 11 $\max|\Delta u|$ will be compared against the user-prescribed maximum allowable nodal temperature change in an increment, $\Delta\theta_{max}$, for controlling the time integration accuracy.
- You must define AMATRX $= K^{NM} + (1/\Delta t)\ C^{NM}$, where C^{NM} is the heat capacity matrix and RHS $= F^N$, and must update the state variables, H^α.

Steady-state coupled thermal-electrical-structural analysis (LFLAGS(1)=102)

- Identical to static analysis, except that the automatic convergence checks are applied to the residuals corresponding to degrees of freedom 1–7, 9, and 11.

Transient coupled thermal-electrical-structural analysis ($\Delta\theta_{max}$) (LFLAGS(1)=103,104)

- Automatic convergence checks are applied to the residuals corresponding to degrees of freedom 1–7, 9, and 11.
- The backward difference scheme is always used for time integration; that is, Abaqus/Standard assumes that $\dot{u}_{t+\Delta t} = \Delta u / \Delta t$, where $\Delta u = u_{t+\Delta t} - u_t$. Therefore, $d\dot{u}/du = 1/\Delta t$ always. For degree of freedom 11 $\max|\Delta u|$ will be compared against the user-prescribed maximum allowable nodal temperature change in an increment, $\Delta\theta_{max}$, for controlling the time integration accuracy.
- You must define AMATRX $= K^{NM} + (1/\Delta t)\ C^{NM}$, where C^{NM} is the heat capacity matrix and RHS $= F^N$; and you must update the state variables, H^α.

Usage with linear perturbation procedures

General and perturbation procedures describes the linear perturbation capabilities in Abaqus/Standard. Here, base state values of variables will be denoted by u^M, H^α, etc. Perturbation values will be denoted by $\tilde{u}^M$, $\widetilde{H}^\alpha$, etc.

Abaqus/Standard will not call user subroutine UEL for the eigenvalue buckling prediction procedure.

For mode-based procedures, user subroutine UEL is called in a prior natural frequency extraction analysis for mass, stiffness, and nondiagonal damping contributions due to damping elements and damping material properties. It is also called to form the left-hand-side matrices and, in some cases, the recovery path.

In addition, for direct-solution and mode-based steady-state dynamic, complex eigenvalue extraction, matrix generation, and substructure generation procedures, Abaqus/Standard calls user subroutine UEL for mass, stiffness, and nondiagonal damping contributions due to damping elements and damping material properties.

Static analysis (LFLAGS(1)=1, 2)

- Abaqus/Standard will solve $K^{NM}\,\tilde{u}^M = \widetilde{P}^N$ for $\tilde{u}^M$, where K^{NM} is the base state stiffness matrix and the perturbation load vector, $\widetilde{P}^N$, is a linear function of the perturbation loads, $\tilde{p}$; that is, $\widetilde{P}^N = (\partial F / \partial \tilde{p})\, \tilde{p}$.

- `LFLAGS(3)=1`: You must define `AMATRX` $= K^{NM}$ and `RHS` $= \widetilde{P}^N$.
- `LFLAGS(3)=100`: You must compute perturbations of the internal variables, $\widetilde{H}^\alpha$, and define `RHS` $= \widetilde{P}^N - K^{NM}\, \tilde{u}^M$ for output purposes.

Eigenfrequency extraction analysis (`LFLAGS(1)=41`)

- $F^N = -M^{NM}\ddot{\tilde{u}} + G^N\left(u^M + \tilde{u}^M, \ldots\right) = -M^{NM}\ddot{\tilde{u}} + \left(\partial G^N / \partial u^M\right) \tilde{u}^M$.
- Abaqus/Standard will solve $K^{NM}\, \phi_i^M \quad = \omega_i^2\, M^{NM}\, \phi_i^M$ for ϕ_i^N and ω_i, where $K^{NM} = -\,\partial F^N / \partial u^M$ is the base state stiffness matrix and $M^{NM} = -\,\partial F^{NM} / \partial \ddot{u}^M$ is the base state mass matrix.
- `LFLAGS(3)=2`: Define `AMATRX` $= K^{NM}$.
- `LFLAGS(3)=4`: Define `AMATRX` $= M^{NM}$.

Direct steady-state analysis (`LFLAGS(1)=95`)

- `LFLAGS(3)=2`: Define stiffness matrix `AMATRX` $= K^{NM}$.
- `LFLAGS(3)=4`: Define mass matrix `AMATRX` $= M^{NM}$.
- `LFLAGS(3)=3`: Define damping matrix `AMATR` $= C^{NM}$.
- `LFLAGS(7)=1`: Define viscous damping matrix `AMATRX` $= C_{viscous}{}^{NM}$.
- `LFLAGS(7)=2`: Define structural damping matrix `AMATRX` $= C_{structural}{}^{NM}$.
- Reaction force (the right-hand vector in a recovery path) is calculated outside the user subroutine. It includes inertia force and forces due to damping. However, you can augment the reaction force with additional contributions when `LFLAGS(3)=2` and `NRHS=2` (denoting the recovery path).

Mode-based dynamic analysis (`LFLAGS(1)=91, 92, 93, 94`)

- `LFLAGS(3)=2`: Define stiffness matrix `AMATRX` $= K^{NM}$.
- `LFLAGS(3)=4`: Define mass matrix `AMATRX` $= M^{NM}$.
- `LFLAGS(3)=3`: Define damping matrix `AMATR` $= C^{NM}$.
- `LFLAGS(7)=1`: Define viscous damping matrix `AMATRX` $= C_{viscous}{}^{NM}$.
- `LFLAGS(7)=2`: Define structural damping matrix `AMATRX` $= C_{structural}{}^{NM}$.
- For these procedures, the user subroutine defines only the left-hand matrices. Right-hand vectors are calculated outside this user subroutine.

Nondiagonal damping in linear perturbation procedures

- For all Abaqus/Standard procedures in which nondiagonal damping can be used, you must set the flag `LFLAGS(3)=3` to indicate that the damping matrix will be provided. To provide the viscous damping matrix, you must also set `LFLAGS(7)=1` and define `AMATRX`$= C_{viscous}{}^{NM}$. To provide structural damping matrix you must also set `LFLAGS(7)=2` and define `AMATRX`$= C_{structural}{}^{NM}$.

Example: Structural and heat transfer user element

Both a structural and a heat transfer user element have been created to demonstrate the usage of subroutine `UEL`. These user-defined elements are applied in a number of analyses. The following excerpt is from the verification problem that invokes the structural user element in an implicit dynamics procedure:

```
*USER ELEMENT, NODES=2, TYPE=U1, PROPERTIES=4, COORDINATES=3,
VARIABLES=12
1, 2, 3
*ELEMENT, TYPE=U1
101, 101, 102
*ELGEN, ELSET=UTRUSS
101, 5
*UEL PROPERTY, ELSET=UTRUSS
```

```
0.002, 2.1E11, 0.3, 7200.
```

The user element consists of two nodes that are assumed to lie parallel to the *x*-axis. The element behaves like a linear truss element. The supplied element properties are the cross-sectional area, Young's modulus, Poisson's ratio, and density, respectively.

The next excerpt shows the listing of the subroutine. The user subroutine has been coded for use in a perturbation static analysis; general static analysis, including Riks analysis with load incrementation defined by the subroutine; eigenfrequency extraction analysis; and direct-integration dynamic analysis. The names of the verification input files associated with the subroutine and these procedures can be found in UEL. The subroutine performs all calculations required for the relevant procedures as described earlier in this section. The flags passed in through the `LFLAGS` array are used to associate particular calculations with solution procedures.

During a modified Riks analysis all force loads must be passed into `UEL` by means of distributed load definitions such that they are available for the definition of incremental load vectors; the load keys U*n* and U*n*NU must be used properly, as discussed in User-defined elements. The coding in subroutine `UEL` must distribute the loads into consistent equivalent nodal forces and account for them in the calculation of the `RHS` and `ENERGY` arrays.

```
      SUBROUTINE UEL(RHS,AMATRX,SVARS,ENERGY,NDOFEL,NRHS,NSVARS,
     1     PROPS,NPROPS,COORDS,MCRD,NNODE,U,DU,V,A,JTYPE,TIME,
     2     DTIME,KSTEP,KINC,JELEM,PARAMS,NDLOAD,JDLTYP,ADLMAG,
     3     PREDEF,NPREDF,LFLAGS,MLVARX,DDLMAG,MDLOAD,PNEWDT,
     4     JPROPS,NJPROP,PERIOD)
C
C#include <cdp.cmn>
      INCLUDE 'ABA_PARAM.INC'
      PARAMETER ( ZERO = 0.D0, HALF = 0.5D0, ONE = 1.D0)
      PARAMETER ( alfdyn = 0.1D0, betdyn=0.02d0, s=0.0D0)
C
      DIMENSION RHS(MLVARX,*),AMATRX(NDOFEL,NDOFEL),
     1     SVARS(NSVARS),ENERGY(8),PROPS(*),COORDS(MCRD,NNODE),
     2     U(NDOFEL),DU(MLVARX,*),V(NDOFEL),A(NDOFEL),TIME(2),
     3     PARAMS(3),JDLTYP(MDLOAD,*),ADLMAG(MDLOAD,*),
     4     DDLMAG(MDLOAD,*),PREDEF(2,NPREDF,NNODE),LFLAGS(*),
     5     JPROPS(*)
      DIMENSION SRESID(6)
C
C UEL SUBROUTINE FOR A HORIZONTAL TRUSS ELEMENT
C
C     SRESID - stores the static residual at time t+dt
C     SVARS  - In 1-6, contains the static residual at time t
C              upon entering the routine. SRESID is copied to
C              SVARS(1-6) after the dynamic residual has been
C              calculated.
C            - For half-increment residual calculations: In 7-12,
C              contains the static residual at the beginning
C              of the previous increment. SVARS(1-6) are copied
C              into SVARS(7-12) after the dynamic residual has
C              been calculated.
C
      AREA = PROPS(1)
      E    = PROPS(2)
      ANU  = PROPS(3)
      RHO  = PROPS(4)
C
      ALEN = ABS(COORDS(1,2)-COORDS(1,1))
      AK   = AREA*E/ALEN
      AM   = HALF*AREA*RHO*ALEN
C
      DO K1 = 1, NDOFEL
        SRESID(K1) = ZERO
        DO KRHS = 1, NRHS
          RHS(K1,KRHS) = ZERO
        END DO
        DO K2 = 1, NDOFEL
          AMATRX(K2,K1) = ZERO
        END DO
      END DO
C
      IF (LFLAGS(3).EQ.1) THEN
C       Static or nonlinear dynamic analysis
C       Normal incrementation
        IF (LFLAGS(1).EQ.1 .OR. LFLAGS(1).EQ.2) THEN
C         *STATIC
          AMATRX(1,1) =  AK
          AMATRX(4,4) =  AK
          AMATRX(1,4) = -AK
          AMATRX(4,1) = -AK
          IF (LFLAGS(4).NE.0) THEN
            FORCE  = AK*(U(4)-U(1))
            DFORCE = AK*(DU(4,1)-DU(1,1))
            SRESID(1) = -DFORCE
            SRESID(4) =  DFORCE
            RHS(1,1) = RHS(1,1)-SRESID(1)
            RHS(4,1) = RHS(4,1)-SRESID(4)
            ENERGY(2) = HALF*FORCE*(DU(4,1)-DU(1,1))
     *           + HALF*DFORCE*(U(4)-U(1))
     *           + HALF*DFORCE*(DU(4,1)-DU(1,1))
          ELSE
            FORCE = AK*(U(4)-U(1))
            SRESID(1) = -FORCE
            SRESID(4) =  FORCE
            RHS(1,1) = RHS(1,1)-SRESID(1)
            RHS(4,1) = RHS(4,1)-SRESID(4)
            DO KDLOAD = 1, NDLOAD
              IF (JDLTYP(KDLOAD,1).EQ.1001) THEN
                RHS(4,1)  = RHS(4,1)+ADLMAG(KDLOAD,1)
                ENERGY(8) = ENERGY(8)+(ADLMAG(KDLOAD,1)
     *               - HALF*DDLMAG(KDLOAD,1))*DU(4,1)
                IF (NRHS.EQ.2) THEN
C                 Riks
                  RHS(4,2) = RHS(4,2)+DDLMAG(KDLOAD,1)
                END IF
              END IF
            END DO
            ENERGY(2) = HALF*FORCE*(U(4)-U(1))
          END IF
        ELSE IF (LFLAGS(1).EQ.11 .OR. LFLAGS(1).EQ.12) THEN
C         *DYNAMIC
          ALPHA = PARAMS(1)
          BETA  = PARAMS(2)
          GAMMA = PARAMS(3)
C
          DADU = ONE/(BETA*DTIME**2)
          DVDU = GAMMA/(BETA*DTIME)
c
          dynt6 = GAMMA/(BETA*DTIME)
          dynt4 = ONE + ALPHA
C
C         LHS operator and RHS: Mass related terms
          DO K1 = 1, NDOFEL
            AMATRX(K1,K1) = AM*DADU + dynt6*dynt4*am*alfdyn
            RHS(K1,1) = RHS(K1,1)-AM*A(K1)
          END DO
C
C         LHS operator: stiffness related terms
          AMATRX(1,1) = AMATRX(1,1)+(ONE+ALPHA)*AK+dynt6*dynt4*betdyn*AK
          AMATRX(4,4) = AMATRX(4,4)+(ONE+ALPHA)*AK+dynt6*dynt4*betdyn*AK
          AMATRX(1,4) = AMATRX(1,4)-(ONE+ALPHA)*AK-dynt6*dynt4*betdyn*AK
          AMATRX(4,1) = AMATRX(4,1)-(ONE+ALPHA)*AK-dynt6*dynt4*betdyn*AK
C         Force
          FORCE = AK*(U(4)-U(1))+(betdyn*AK)*(v(4)-v(1))
          SRESID(1) = -FORCE+alfdyn*AM*v(1)
          SRESID(4) =  FORCE+alfdyn*AM*v(4)

C         RHS due to stiffnes and damping
          RHS(1,1) = RHS(1,1) -
     *         ((ONE+ALPHA)*SRESID(1)-ALPHA*SVARS(1))
          RHS(4,1) = RHS(4,1) -
     *         ((ONE+ALPHA)*SRESID(4)-ALPHA*SVARS(4))
          ENERGY(1) = ZERO
          DO K1 = 1, NDOFEL
            SVARS(K1+6) = SVARS(k1)
            SVARS(K1)   = SRESID(K1)
            ENERGY(1)   = ENERGY(1)+HALF*V(K1)*AM*V(K1)
          END DO
C
          ENERGY(2) = HALF*AK*(U(4)-U(1))*(U(4)-U(1))
        END IF
C

      ELSE IF (LFLAGS(3).EQ.2) THEN
```

```
C          Stiffness matrix requested
           DO K1 = 1, NDOFEL
            AMATRX(K1,K1) = AK
           END DO
           AMATRX(1,4) = -AK
           AMATRX(4,1) = -AK
      ELSE IF (LFLAGS(3).EQ.3) THEN
C       Damping requested
        if(lflags(7).eq.1) then
C          Viscous damping matrix requested
C          Mass matrix diagonal only
           DO K1 = 1, NDOFEL
            AMATRX(K1,K1) = betdyn*AK+alfdyn*AM
           END DO
           AMATRX(1,4) = -betdyn*AK
           AMATRX(4,1) = -betdyn*AK
        else if(lflags(7).eq.2) then
C          Structural damping matrix requested
           DO K1 = 1, NDOFEL
            AMATRX(K1,K1) = s*AK
           END DO
           AMATRX(1,4) = -s*AK
           AMATRX(4,1) = -s*AK
        end if
      ELSE IF (LFLAGS(3).EQ.4) THEN
C       Mass matrix
        DO K1 = 1, NDOFEL
          AMATRX(K1,K1) = AM
        END DO
      ELSE IF (LFLAGS(3).EQ.5) THEN
C       Half-increment residual calculation
        ALPHA = PARAMS(1)
        FORCE = AK*(U(4)-U(1))
        SRESID(1) = -FORCE
        SRESID(4) =  FORCE
        RHS(1,1) = RHS(1,1)-AM*A(1)-(ONE+ALPHA)*SRESID(1)
     *       + HALF*ALPHA*( SVARS(1)+SVARS(7) )
        RHS(4,1) = RHS(4,1)-AM*A(4)-(ONE+ALPHA)*SRESID(4)
     *       + HALF*ALPHA*( SVARS(4)+SVARS(10) )
      ELSE IF (LFLAGS(3).EQ.6) THEN
C       Initial acceleration calculation
        DO K1 = 1, NDOFEL
          AMATRX(K1,K1) = AM
        END DO
        FORCE = AK*(U(4)-U(1))
        SRESID(1) = -FORCE
        SRESID(4) =  FORCE
        RHS(1,1) = RHS(1,1)-SRESID(1)
        RHS(4,1) = RHS(4,1)-SRESID(4)
        ENERGY(1) = ZERO
        DO K1 = 1, NDOFEL
          SVARS(K1) = SRESID(K1)
          ENERGY(1) = ENERGY(1)+HALF*V(K1)*AM*V(K1)
        END DO
        ENERGY(2) = HALF*FORCE*(U(4)-U(1))
      ELSE IF (LFLAGS(3).EQ.100) THEN
C       Output for perturbations
        IF (LFLAGS(1).EQ.1 .OR. LFLAGS(1).EQ.2) THEN
C         *STATIC
          FORCE  = AK*(U(4)-U(1))
          DFORCE = AK*(DU(4,1)-DU(1,1))
          SRESID(1) = -DFORCE
          SRESID(4) =  DFORCE
          RHS(1,1) = RHS(1,1)-SRESID(1)
          RHS(4,1) = RHS(4,1)-SRESID(4)
          ENERGY(2) = HALF*FORCE*(DU(4,1)-DU(1,1))
     *         + HALF*DFORCE*(U(4)-U(1))
     *         + HALF*DFORCE*(DU(4,1)-DU(1,1))
          DO KVAR = 1, NSVARS
            SVARS(KVAR) = ZERO
          END DO
          SVARS(1) = RHS(1,1)
          SVARS(4) = RHS(4,1)
        ELSE IF (LFLAGS(1).EQ.41.or.LFLAGS(1).EQ.47) THEN
C         *FREQUENCY or *COMPLEX FREQUENCY
          DO KRHS = 1, NRHS
```

```
            DFORCE = AK*(DU(4,KRHS)-DU(1,KRHS))
            SRESID(1) = -DFORCE
            SRESID(4) =  DFORCE
            RHS(1,KRHS) = RHS(1,KRHS)-SRESID(1)
            RHS(4,KRHS) = RHS(4,KRHS)-SRESID(4)
          END DO
          DO KVAR = 1, NSVARS
            SVARS(KVAR) = ZERO
          END DO
          SVARS(1) = RHS(1,1)
          SVARS(4) = RHS(4,1)
        END IF
      END IF
C
      RETURN
      END
```