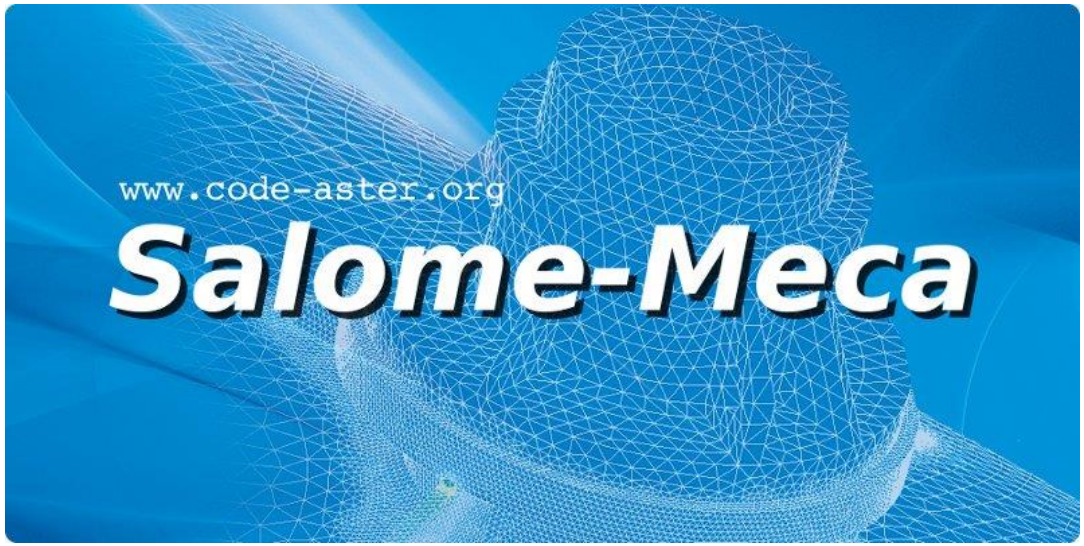
Non-linear analysis – Advanced topics



Code_Aster, Salome-Meca course material

Outline

- The DEFI_LIST_INST command
- Multi-step analysis (reuse and initial state)
- Advanced algorithms controls
- Real-time monitoring

The DEFI_LIST_INST command:

More control on non-linear analysis

More control on non-linear analysis

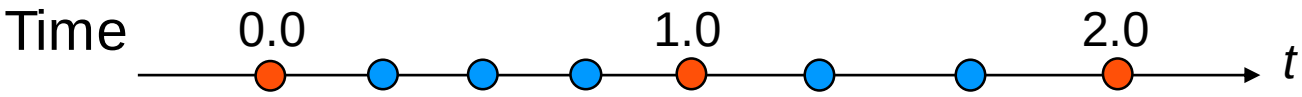
- When to use `DEFI_LIST_INST` ?
  - For automatic time-step control
  - Event-driven system

The DEFI_LIST_INST command:

Advanced time-step control

Advanced time-step control

- Non-linear problem is parametrized by time – Basic




```
L_REAL=DEFI_LIST_REEL( DEBUT=0.0,
                       INTERVALLE=(
                           _F(JUSQU_A=1.0,NOMBRE=4,),
                           _F(JUSQU_A=2.0,NOMBRE=3,),
                       ),)
```

```
RESUN = STAT_NON_LINE(...
                          INCREMENT=_F(LIST_INST=L_REAL)
                          ...)
```

Deprecated in V13.3 => use `DEFI_LIST_INST` instead !

Advanced time-step control

- Non-linear problem is parametrized by time – Advanced

Initial time list defined by `DEFI_LIST_REEL`

```
L_REAL = DEFI_LIST_REEL(
             DEBUT=0.0,
             INTERVALLE=(_F(JUSQU_A=2.0,NOMBRE=1,),),)
```

```
DEFLIST = DEFI_LIST_INST(
             DEFI_LIST = _F(LIST_INST = L_REAL,),)
```

```
RESUN = STAT_NON_LINE(...
                      INCREMENT=_F(LIST_INST= DEFLIST)
                      ...)
```

Advanced time-step control

- Time-step control – Manual mode:
  - From user's defined list of values: `DEFI_LIST_REEL`

```
DEFLIST = DEFI_LIST_INST(
              DEFI_LIST = _F(MODE ='MANUEL',
                             LIST_INST = L_REAL,),)
```

  - From user's defined list of values: Python's list in `VALE`

```
DEFLIST = DEFI_LIST_INST(
              DEFI_LIST = _F(MODE ='MANUEL',
                             VALE = (1.,2.,3.,),)
```

Advanced time-step control

- Time-step control – Manual mode
  - From previous computation:
    - First computation with automatic mode for instance => results `RESU1`
    - Second computation with previous list, divided by two for instance (`SUBD_PAS=2`)

```
DEFLIST = DEFI_LIST_INST(
              DEFI_LIST = _F(MODE ='MANUEL',
                             RESULTAT = RESU1,
                             SUBD_PAS = 2),)
```

Advanced time-step control

- Time-step control – Automatic mode
  - Definition of global parameters: minimum and maximum step, maximum number of steps

```
DEFLIST = DEFI_LIST_INST(
              DEFI_LIST = _F(MODE ='AUTO',
                             PAS_MINI = 1.E-3,
                             PAS_MAXI = 0.1
                             NB_PAS_MAXI = 1000.),)
```

- `PAS_MINI` reached => stop
- `PAS_MAXI` reached => time step is capped

Advanced time-step control

- Time-step control – Automatic mode
  - Definition of automatic adaptation schemes in `ADAPTATION` keyword: how to compute next time step ?

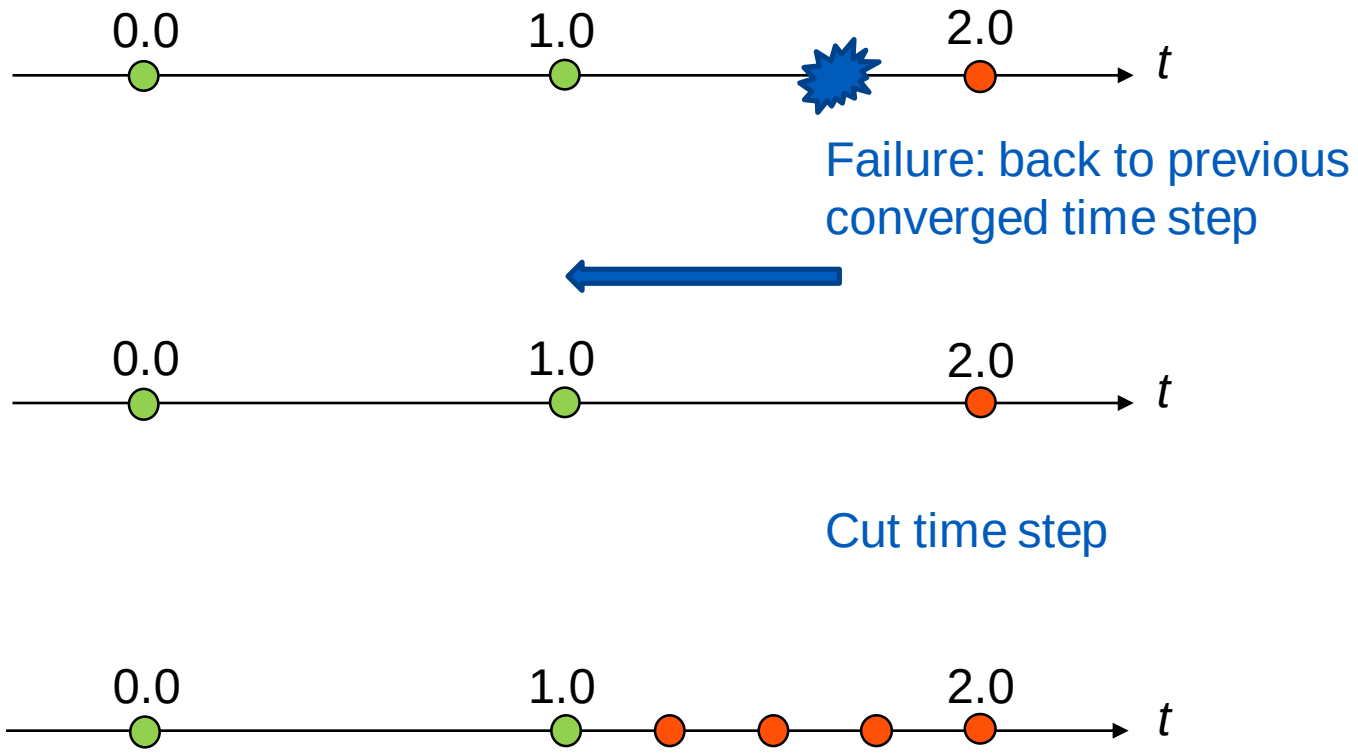
$$\Delta t^{i+1} = c.\,\Delta t^{i} \ ?$$

  - **Event-driven system: `EVENEMENT+MODE_CALCUL_TPLUS`**

```
DEFLIST = DEFI_LIST_INST(
            DEFI_LIST  = _F(MODE ='AUTO',...),
            ADAPTATION = _F(EVENEMENT ='SEUIL',
                  MODE_CALCUL_TPLUS = 'FIXE',
                  PCENT_AUGM = 50.,),)
```

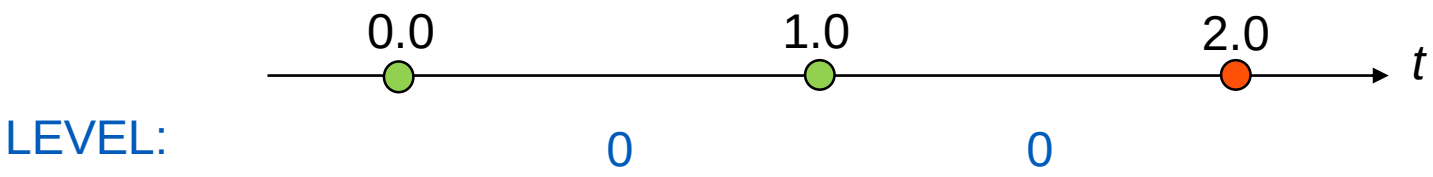
Advanced time-step control

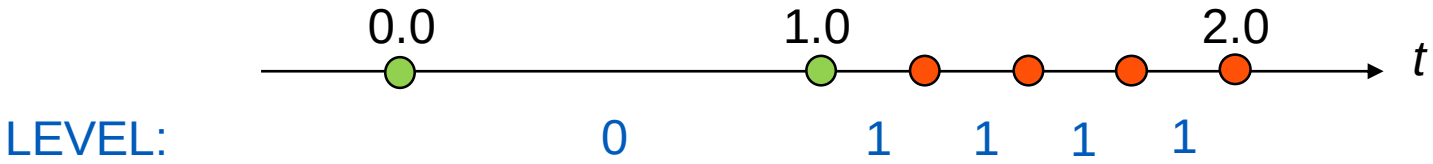
- Time-step control – Manage step time
  - **Event-driven system:** `EVENEMENT='ERREUR' +ACTION='DECOUPE`



Advanced time-step control

- Time-step control – The LEVEL
  - Level of time step: initial from DEFI_LIST_REEL is **LEVEL 0**





LEVEL = **NIVEAU**

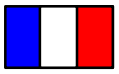
LEVEL is recursive …

The DEFI_LIST_INST command:

Events management

Events management

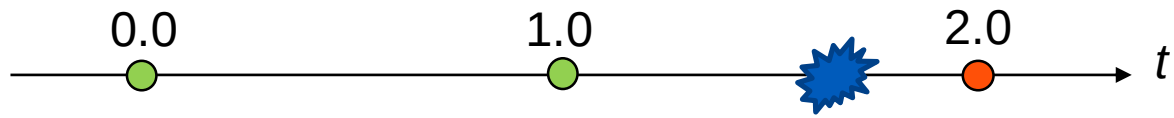
- General event mechanism
  - Event-driven is how to control algorithm: stop computation, manage step time, other actions
  - Direct event-driven: keywords **EVENEMENT** and **ACTION**
  - Time step adaptation event-driven: keywords **ADAPTATION** and **MODE_CALCUL_TPLUS**



```
EVENT          = EVENEMENT
ACTION         = ACTION
EVENT+ACTION   = ECHEC
```

Events management

- The EVENT and the ACTION: an example for step time management



EVENT: ERROR (ex.: failure in constitutive equation solving, too many Newton iterations, etc)

ACTION: Cut time step

Events management

- List of the EVENEMENT for general event management
- ERREUR: error during constitutive equation solving, too many Newton iterations, contact error, singular matrix
- DIVE_RESI: residual has increased twice during Newton iterations
- DELTA_GRANDEUR: the increase of a given quantity has exceeded a value defined by the user (ex: cumulative plastic strain)
- COLLISION: contact occurs for the first time
- INTERPENETRATION: contact interpenetration (for penalty methods)
- INSTABILITE: detection of an instability (CRIT_STAB operator)

Some of these events have parameters (see U4.34.04). Example:

```
_F(EVENEMENT= 'DELTA_GRANDEUR',
   VALE_REF = 0.1e-2,
   NOM_CHAM = 'VARI_ELGA',
   NOM_CMP  = 'V1'),
```

For
VMIS_ISOT_LINE
$\Delta\varepsilon_{eq}^{p} > 0.1\%$

Events management

- List of ACTION for general event management
- ARRET: stop computation (previous converged steps are saved)
- DECOUPE: cut time step
- ITER_SUPPL: allow more Newton iterations (more than ITER_GLOB_MAXI parameter)
- ADAPT_COEF_PENA: adaptation of penalty parameter for contact
- CONTINUE: continue computation

Some of these actions have parameters (see U4.34.04).

Events management

- List of the EVENEMENT for AUTO time stepping
- SEUIL: defining the trigger for adapting the time step in automatic mode
- TOUT_INST: adaptation occurs at every step time

Some of these events have parameters (see U4.34.04). Example:

```
_F(EVENEMENT= 'SEUIL',
   NB_INCR_SEUIL=2,
   NOM_PARA='NB_ITER_NEWTON',
   CRIT_COMP='LE',
   VALE_I=5),
```

= If two successive time steps use less than 5 Newton iterations

Events management

- List of the MODE_CALCUL_TPLUS for AUTO time stepping

What is c in $\Delta t = c.\ \Delta t^i$?

- FIXE: c is constant (PCENT_AUGM parameter)
- DELTA_GRANDEUR: c is defined by the user from quantities computed (displacement, stress…)
- IMPLEX : c is automatically defined for IMPLEX
- ITER_NEWTON : c is defined by the variation of the number of Newton iterations during time stepping

Some of these actions have parameters (see U4.34.04).

The DEFI_LIST_INST command:

Summary

The DEFI_LIST_INST command – Summary

- By default: at least one event (=ERREUR) with its action (=DECOUPE)
- You can define more than one ECHEC (event and its action)
- Yan can define more than one ADAPTATION (the smallest time step time is selected)
- Advanced users may define special heuristics for time adaptation or event-driven => these may then be integrated in the official version of Code_Aster

Multi-step analyis

Multi-step analysis

- What can I change between two non-linear computations ?
  - **Model** from `AFFE_MODELE`
    - Pay attention to the initial state if you change the model !
  - **Loads** from `AFFE_CHAR_*` and `EXCIT`
  - **Elementary characteristics** from `AFFE_CARA_ELEM`
  - **Mechanical behaviours** from `COMPORTEMENT` keyword:
    - From ELASTIC to non-linear behavior laws => OK !
    - From a non-linear behavior laws to another => Not OK (because of internal state variables)

You cannot change the mesh from one computation to another :
➔ you have to project results (use PROJ_CHAMP with ECLA_PG for Gauss fields)
➔ you have to re-define everything (models, loads, …)

Note : for excavation or layer-based structures, see U2.04.06

Multi-step analysis

- Initial state is defined by:
  - Displacements
  - Stresses
  - Internal state variables
  - +more specific fields for certain applications such as XFEM or multifiber beam elements
- By default: initial state is empty (everything is zero)
- The initial index to access data fields (`NUME_ORDRE`) is always 0 except when the result datastructure was extended (`reuse`) => CANNOT BE CHANGED BY THE USER
- The initial time may be defined by `ETAT_INIT/INST_ETAT_INIT` keyword if necessary
- Careful ! In `reuse` mode, `INST_ETAT_INIT` must be superior to the previous time step

Multi-step analysis

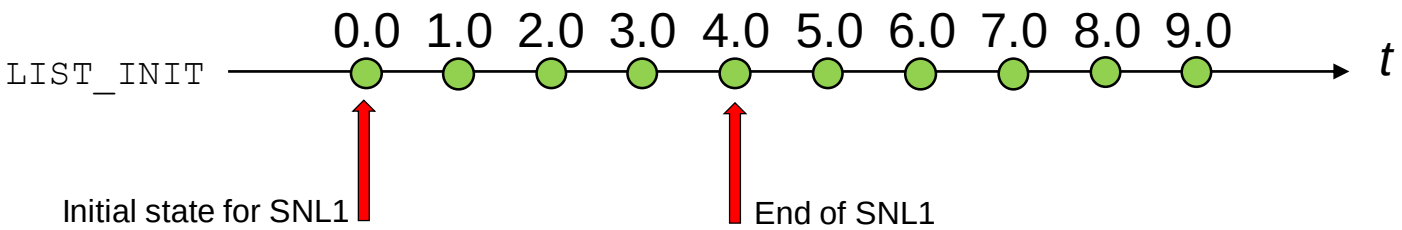
- Defining a specific initial state by `ETAT_INIT` keyword
  - From previous `STAT_NON_LINE`: `EVOL_NOLI` + `NUME_ORDRE` or `INST` or `{nothing}`
    - If `{nothing}` => last computed step in datastructure
  - From individual fields or manually assembled datastructures
    - Field by field: displacement (`DEPL`), stress (`SIGM`), internal state variables (`VARI`)
    - Complete datastructure: from `LIRE_RESU` or `CREA_RESU` command

Multi-step analysis

- For non-linear behavior laws:
  - The initial stress field is used to update stresses during behavior law integration
  - Depend on laws: some of them (hyperelastic for large strains) DON'T take into account initial stresses => you should use internal state variables (`SIMO_MIEHE` for instance)
  - For `GDEF_LOG` strains measure => stresses are not Cauchy but logarithmic ones (see ssnp159b for instance)
  - To have initial strains: use `AFFE_CHAR_MECA/PRE_EPSI`
  - To have initial stresses that are not used to update stresses (*e.g. during behavior law integration)*: use `AFFE_CHAR_MECA/PRE_SIGM`
- For contact (`CONTINUE` formulation):
  - Initial contact behavior is activated by default in `DEFI_CONTACT` command (`CONTACT_INIT='INTERPENETRE'`)
  - Friction history is constructed from displacement field (`DEPL`) => careful !

Multi-step analysis

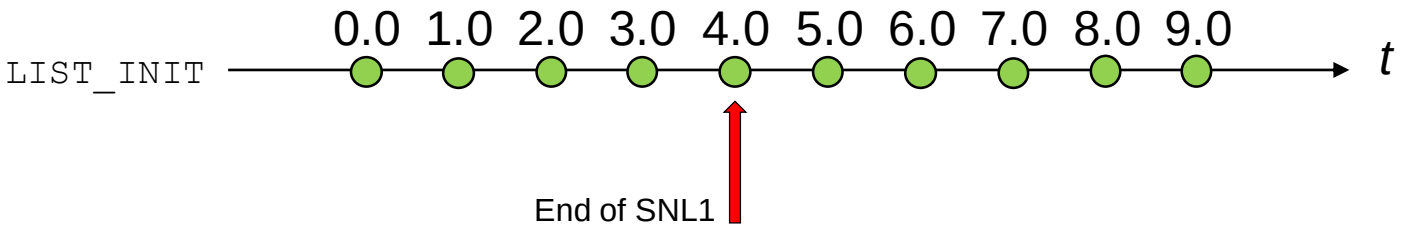
- Management of time list:
  - Defined in `INCREMENT/LIST_INST`
  - You can manage the start/end of the time list with `NUME_INST_INIT / INST_INIT / NUME_INST_FIN / INST_FIN`
  - To end at `INST=4.0`



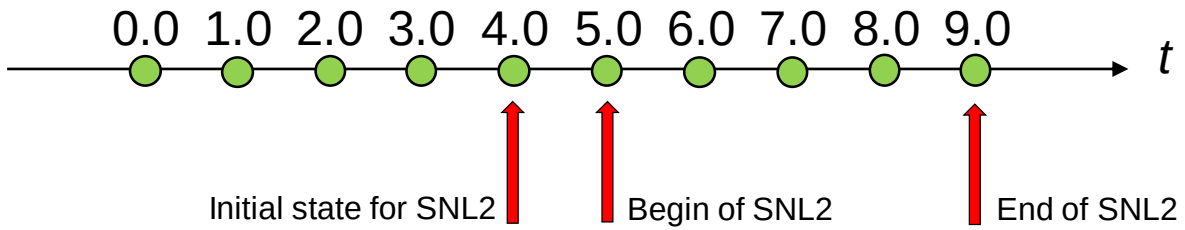

```
SNL1 : INCREMENT = _F(LIST_INST = LIST_INIT, INST_FIN = 4.0)
```

Multi-step analysis

- Management of time list:
  - « Simple » continuation of a non-linear computation




```
SNL1 : INCREMENT = _F(LIST_INST = LIST_INIT, INST_FIN = 4.0)
```

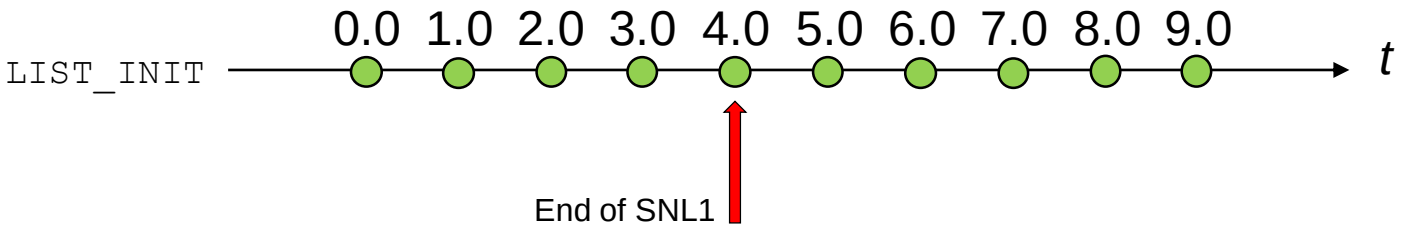


Standard reuse:

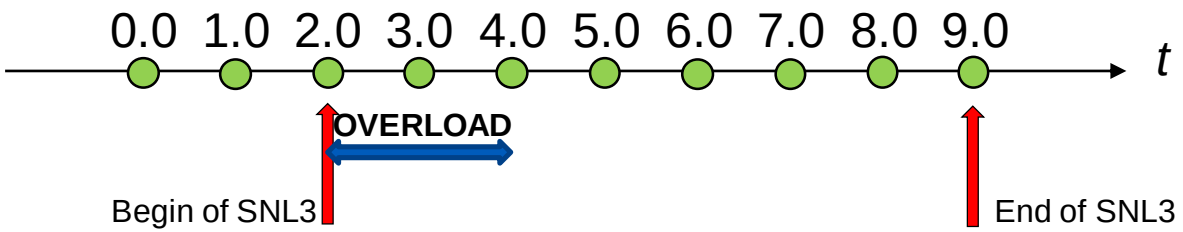
```
SNL2 : reuse = SNL1
       ETAT_INIT = _F(EVOL_NOLI=SNL1),
       INCREMENT = _F(LIST_INST = LIST_INIT)
```

Multi-step analysis

- Management of time list:
  - Overload a non-linear computation




```
SNL1 : INCREMENT = _F(LIST_INST = LIST_INIT, INST_FIN = 4.0)
```

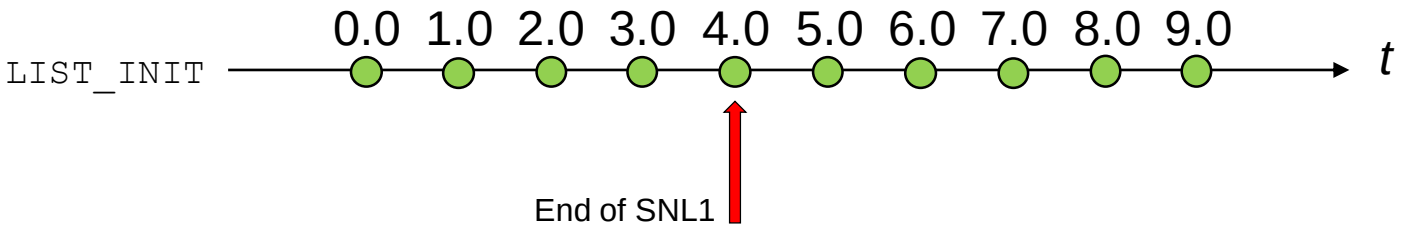


Overload :

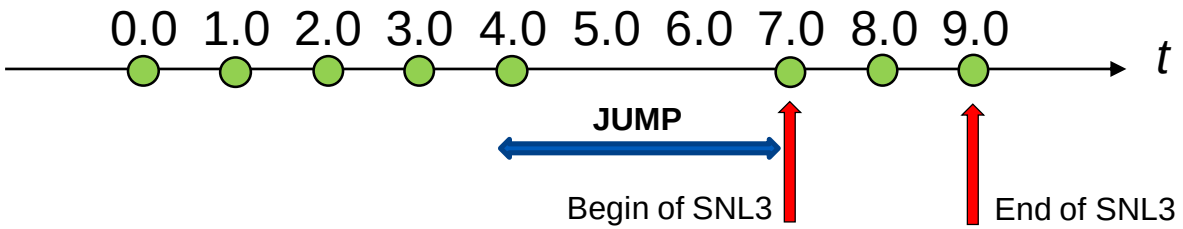
```
SNL3 : reuse = SNL1
       ETAT_INIT = _F(EVOL_NOLI=SNL1),
       INCREMENT = _F(LIST_INST = LIST_INIT, INST_INIT = 2.0)
```

Multi-step analysis

- Management of time list:
  - Jump over a non-linear computation




```
SNL1 : INCREMENT = _F(LIST_INST = LIST_INIT, INST_FIN = 4.0)
```



Jump :

```
SNL4 : reuse = SNL1
       ETAT_INIT = _F(EVOL_NOLI=SNL1),
       INCREMENT = _F(LIST_INST = LIST_INIT, INST_INIT = 7.0)
```

Advanced algorithm controls

Advanced algorithm controls

- Global non-linear solver – Keyword `METHODE`
  - Newton and quasi-Newton method – `METHODE = 'NEWTON'`
  - Implicit/explicit method (Oliver and *al.*) – `METHODE = 'IMPLEX'`
    - For damaged concrete analysis (to use with special time step adaptation)
  - Newton-Krylov method – `METHODE = 'NEWTON_KRYLOV'`
    - To use with an iterative linear solver (`PETSC` or `GCPC`), adaptative convergence criterion in the iterative solver ➔ very efficient if a good preconditionner is available

Advanced algorithm controls

- Newton solver – Keyword `METHODE='NEWTON'`
  - Quasi-Newton options: see `REAC_INCR` and `REAC_ITER` to update matrix
  - Choice of matrix for prediction phase (Euler):
    - `PREDICTION='ELASTIQUE'/'TANGENTE'` => prefer `PREDICTION='ELASTIQUE'` for severe non-linear analysis (especially contact) and unloading
    - `PREDICTION='DEPL_CALCULE'` => useful to run post-critical analysis or damage
  - Choice of matrix for correction phase (Newton):
    - `MATRICE='TANGENTE'` => exact tangent matrix (if available from the constitutive law), should generally be preferred (default)
    - `MATRICE='ELASTIQUE'` => will converge for generalized material (very slowly though)
    - To swap from `TANGENTE` to `ELASTIQUE` => use `PAS_MINI_ELAS`
    - For damaged structure with elastic matrix => use `REAC_ITER_ELAS`
    - Force symmetric matrix with `MATR_RIGI_SYME='OUI'`

Advanced algorithm controls

- Storage management (`ARCHIVAGE` keyword)
  - Default:
    - All time steps are saved with displacements, stresses and internal state variables
  - User-defined storage:
    - Use `ARCHIVAGE + LIST_INST/INST/PAS_ARCH` to store only part of the results. Initial state and last converged step are **ALWAYS** saved (to allow restart after errors, lack of CPU time for instance)
    - Use `ARCHIVAGE + CHAM_EXCLU` to exclude some fields in the saving process
    - Warning: saving all fields for a lot of time steps and a large model => VERY LARGE MEMORY

Advanced algorithm controls

- Output management (`AFFICHAGE` keyword)
  - The convergence table is printed at each time step
  - To save this table in a CSV file => `AFFICHAGE/UNITE`
  - To print only for some steps (useful when there's a lot of time steps) => `AFFICHAGE/PAS`
  - To print where residuals are maximum in convergence table => `AFFICHAGE/INFO_RESIDU='OUI'`
  - To print CPU time for each Newton iteration => `AFFICHAGE/INFO_TEMPS='OUI'`

Advanced algorithm controls

- Statistics management (`MESURE` keyword)
  - Some statistics (CPU time, number) are printed at each time step and at the end of computation
  - To save this table in a CSV file => `MESURE/UNITE`
  - To generate a TABLE in Aster format => `MESURE/TABLE='OUI'`

```
RESU = STAT_NON_LINE(
    MESURE = _F(TABLE='OUI',UNITE=50),)
STAT = RECU_TABLE(CO=RESU, NOM_TABLE='STAT',)
IMPR_TABLE(TABLE=STAT)
```

Real-time monitoring

Real-time monitoring

- Extract values for monitoring analysis (OBSERVATION/SUIVI_DDL keywords)
  - Select field:
    - DEPL, VITE, ACCE, TEMP
    - SIEF_ELGA, VARI_ELGA, EPSI_ELGA
    - CONT_NOEU
    - FORC_NODA, REAC_NODA
    - *_ABSOLU
  - Select where:
    - TOUT/GROUP_MA/MAILLE for ELEM fields + Gauss point / sub-point
    - TOUT/GROUP_MA/MAILLE/GROUP_NO/NOEUD for NODE fields
  - Select what:
    - Value of component (ex. DX, DY, SIXX, V1, etc.) or formula (ex. sqrt($DX^2+DY^2+DZ^2$))
    - On selected or global on several (ABS, MAX, MIN, …)

Real-time monitoring

- Save values in TABLE (`OBSERVATION` keyword)
  - Extract (field, where, what)
  - Which step time: `OBSERVATION+LIST_INST/INST/PAS_OBSE`
  - Save in TABLE
  - Get from results

```
RESU = STAT_NON_LINE(
    OBSERVATION = _F(NOM_CHAM='SIEF_ELGA',
                     EVAL_CMP='VALE',
                     NOM_CMP='M11',
                     EVAL_ELGA='MAX',
                     TOUT='OUI'),)

OBSV = RECU_TABLE(CO=RESU, NOM_TABLE='OBSERVATION',)

IMPR_TABLE(TABLE=OBSV)
```

Real-time monitoring

- Print values in message file (`SUIVI_DDL` keyword)
  - Extract (field, where, what)
  - Print in message file (convergence table)
  - Warning: number of `SUIVI_DDL` is limited (only 15 columns in convergence table)
  - As is convergence table => save in message file (`AFFICHAGE` keyword) in real-time

```
RESU = STAT_NON_LINE(
    SUIVI_DDL=(_F(NOM_CMP='PRE1',
                  NOM_CHAM='DEPL',
                  EVAL_CHAM='MAXI_ABS',
                  NOEUD=('NO1'),),)
```

End of presentation

Is something missing or unclear in this document?

Or feeling happy to have read such a clear tutorial?

Please, we welcome any feedbacks about Code_Aster training materials.

Do not hesitate to share with us your comments on the Code_Aster forum dedicated thread.