Productivity and Quality of LS-DYNA® Analyses: Implementing a Tailor-made Software Solution for the Transport and Storage of Radioactive Materials

Gilles Marchaud, Valérie Saint-Jean
ORANO TN, Montigny-le-Bretonneux, France

Abstract

For more than 50 years, ORANO TN (formerly AREVA TN) has been supplying customer-focused, innovative transportation and storage solutions for radioactive material with the highest levels of safety and security.

Within a context of stringent regulations, ORANO TN performs LS-DYNA analyses to evaluate the crashworthiness of casks and to reduce the number of costly real tests. Continuous effort is being made to improve these analyses. Part of the effort is dedicated to Verification & Validation, i.e. ensuring that LS-DYNA, along with well-defined methodologies, provides realistic results.

Considerable effort is also being made to improve Productivity and Quality, with the aim of bringing more value to our customers. Time-consuming, error-prone tasks have been identified in the whole analysis process: pre-processing, post-processing, model checking, report writing and checking.

LS-PrePost® command files are already used by FE analysts on a day-to-day basis. Perl and Python scripts perform specific tasks such as: low-pass filtering, stress linearization in bolt shanks, determination of extrema in xy-plot files,…

However, the need has arisen for a more comprehensive tool. For instance, when plotting the effective plastic strain contours on a wooden part involved in energy absorption, the analyst has to manually set the fringe range bounds in order to check that the part is not overly compacted. The information used for these bounds is the one used in the early phase of pre-processing, when defining material properties. More generally, any data relevant in a given phase of the analysis process is likely to be relevant in the other phases.

*Consequently, for more than two years, a Python library has been developed. Being object-oriented, it gives easy access to keywords. It checks that *CONTROL parameters are set according to the best practice. It sets material properties from engineering values. It performs advanced post-processing tasks on ASCII and binary output files.*

The present paper will be illustrated with selected features of the Python library.

Introduction

Stringent safety regulations, such as [1], are applicable to the transportation or storage of radioactive material in the best safety and security conditions.

ORANO TN performs LS-DYNA [2] analyses to evaluate the crashworthiness of casks and to reduce the number of costly real tests.

Continuous effort is being made to improve these analyses.

Part of the effort is dedicated to Verification & Validation, i.e. ensuring that LS-DYNA, along with well-defined methodologies, provides realistic results. Our approach has been presented in [3].

Considerable effort is also being made to improve Productivity and Quality, with the aim of bringing more value to our customers. Our approach is presented below.

Analyzing the FEA tasks in terms of productivity and quality

Time-consuming, error-prone tasks have been identified in the whole FE analysis process:

- pre-processing,
- post-processing,
- model checking,
- report writing and checking.

Typical time-consuming and/or error-prone task items are listed in the following non-exhaustive table:

Task	Item	Goal / Question to be answered
Pre-processing	Dimensions/geometry	Generate a solid/shell mesh on a CAD model.
	Material data input	Convert specified engineering properties into true properties.
	Definition of *CONTROL parameters	Define *CONTROL_PARALLEL with the recommended values for SMP consistency.
Post-processing	Energy-balance graphs	Create energy-balance graphs for 12 parts and 8 cases.
	Acceleration graphs	Create acceleration graphs for 3 accelerometers, 2 cut-off frequencies and 8 cases.
	Contour plots	Create contour plots of maximum equivalent plastic strain for 12 parts and 8 cases, with fringe range bounds adapted according to the material type, e.g. an upper bound equal to the ultimate strain for metals, a lower bound equal to the compaction threshold for woods.

(Table to be continued…)

(Table continued from previous page)

Task	Item	Goal / Question to be answered
Model checking	Dimensions	Do mesh dimensions match those in specified drawings?
	Material data input	Are true properties consistent with specified engineering properties?
	Definition of *CONTROL parameters	Has *CONTROL_PARALLEL been defined with the recommended values for SMP consistency?
	Good convergence	Is the energy ratio close to 1? Is the contact energy (excluding friction) small compared to the maximum internal energy?
Report writing and checking	Material-to-part assignment	The analyst wants to insert figures showing which material is assigned to which part.
	Section-to-part assignment	The analyst wants to insert figures showing which thickness is assigned to which shell part.
	Value tables	The final value of equivalent plastic strain has to be written down for 12 parts and 8 cases.
	Insertion of figures	Did the analyst insert the energy-balance graph for case #3 and not for another case?

The need for automation originates from the need for quality and productivity and is strongly linked with human and organizational factors:

Quality / Productivity	Item	What automation brings to the item	Human and Organizational Factors
Quality	Traceability	The operations are recorded and can therefore be checked afterward.	The checker is not limited to checking the final model/results. The checker can also check the steps leading to them.
	Repeatability	The operations are recorded and can therefore be repeated afterward with perfect fidelity.	Correctly performing a sequence of manual operations does not mean you will perform them correctly once again, and you will need to pay attention to each operation.
	Correctibility / Continuous improvement	The operations are recorded and can therefore be repeated afterward, after some corrections.	Should the model need to be corrected manually, the analyst corrects it and then only needs to re-run the automated post-processing task. The analyst limits their efforts to the steps that need to be corrected.
Productivity	Speed	The operations are recorded and can therefore be repeated afterward on demand.	Once the analyst has launched the appropriate sequence of operations, the analyst does not need to intervene.
	Reusability / Adaptability / Know-how capitalization	The operations are recorded and can therefore be repeated afterward for similar cases, potentially with some modifications.	The analyst limits their efforts to what needs to be modified. The analyst can capitalize on already automated operations by integrating them in a more complex task.

Automation: different ways, with pros and cons

Automation can be implemented in several ways such as:

- Pre-programmed treatment driven by a dialog box in a Graphical User Interface (GUI)
- “Purely sequential sequence” of operations (i.e. without loops, conditional constructs such as “if then else”, …) listed in a command file, read by a GUI (e.g. LS-PrePost command files)
- Non-interactive program developed in a compiled language (e.g. C++ or Fortran)
- Non-interactive script developed in an interpreted language (e.g. Perl or Python).

The differences between a GUI and a program or script are mainly:

GUI vs. Program/script	Pros	Cons
GUI (*)	User-friendliness	Usually proprietary and not open source => more or less adaptable by users
Program/script	Highly adaptable to users’ needs	Not interactive Requires programming skills

(*) A GUI may have the pros of programs/scripts when it can be driven by a script.
Advanced programming is possible if the GUI proposes a comprehensive API
(Application Programming Interface), i.e. one that gives access to all features of the GUI.

The user-friendliness of a GUI is unparalleled:

- for exploring an unknown model
- for building a model from scratch
- for having a quick look at the model results (convergence, general behavior)
- for performing some post-processing task for the first time (dialog boxes are helpful for that).

Programs and scripts are very efficient for well-defined tasks that are repeated many times.

The differences between a (compiled) program and an (interpreted) script are mainly:

Program vs. Script	Pros	Cons
(Compiled) program	High speed Low memory usage	Many code lines Slow debugging
(Interpreted) script	Compact code Fast debugging (*)	Low speed High memory usage

(*) A normal run is in fact performed in debug mode, in such a way that
debugging is even possible when the script is exploited in production phase.

The low speed of scripts can be mitigated by encapsulating CPU-intensive operations in compiled programs. An interpreted language like Python can be extended with modules developed in C++.

From low-scale automation to high-scale automation

LS-PrePost [4] command files are already used by ORANO TN's FE analysts on a day-to-day basis.

Scripts in Perl [5] and Python [6] perform specific tasks such as:

- low-pass filtering,
- stress linearization in bolt shanks,
- determination of extrema in xy-plot files,…

However, the need has arisen for a more comprehensive tool.

For instance, when plotting the effective plastic strain contours on a wooden part involved in energy absorption, the analyst has to manually set the fringe range bounds in order to check that the part is not overly compacted. The information used for these bounds is the one used in the early phase of pre-processing, when defining material properties.

More generally, any data relevant in a given phase of the analysis process is likely to be relevant in the other phases.

LS-PrePost Scripting Command Language (SCL) was tested successfully for a CPU-intensive task: morphing a large mesh by applying a complex mapping of nodal coordinates. The process was fast and the result was immediately visible in the 3D graphics window. However, this C-like computer language was discarded due to its lack of object-orientedness, of complex data structures such as variable-size lists or dictionaries (≈ arrays with string indices), which would have compromised the expected development of increasingly complex tasks.

Consequently, for more than two years, a Python library named "PyDyna" has been developed by ORANO TN.

Being object-oriented, it gives easy access to keywords. It checks that *CONTROL parameters are set according to the best practice. It sets material properties from engineering values. It performs advanced post-processing tasks on ASCII and binary output files.

Some features are similar to those implemented in LS-DYNA-Tools (see DYNAmore web site [7]).

Selected features of "PyDyna" are presented below.

Examples of “PyDyna” features

The following table shows the excerpts of a Python script that uses “PyDyna” for both checking a model and post-processing the results of 3 calculation cases. Even though the Python language is self-explanatory, the script has been completed with comments in **green**:

(1/5) Excerpts from the Python script using “PyDyna”

```
# -*- coding: cp1252 -*-
# +------------------------------------------------------+
# | Copyright (c) ORANO TN, 2018 - All Rights Reserved. |
# +------------------------------------------------------+
import sys              # System-specific parameters and functions
sys.path.append("<<path where pydyna is located>>")
import pydyna_2018_01_01 as pydyna
#--------------------------------------------------------------------------------
# +------------------+
# | Import modules |
# +------------------+
import os               # Miscellaneous operating system interfaces
import math             # Mathematical functions
# (...)
#--------------------------------------------------------------------------------

current_working_directory = os.getcwd()
main_dir  = "."
sub_dirs = [ "Case1", "Case2", "Case3" ]

for sub_dir in sub_dirs: # Scan all sub-directories where cases have been run
    model1 = pydyna.C_LSDynaModel() # Instantiate an object of class “CLSDynaModel”
    #
    model1.set_language("EN") # Figures will be created with English text
    #
    post_dir_for_case = "POST_%s" % sub_dir
    model1.define_post_directories(post_dir_for_case)
    create_directory_if_absent(model1.path_for_post)
    #
    model1.log_print("####  " + sub_dir) # Print sub-directory name in log file
    #
    model_dir = os.path.join(main_dir,sub_dir)
    #
    file_name_k = "model.k"
    #
    if True: # Read keyword file
        bool_skip_nodes_elements = False # Skipping these would accelerate reading
        bool_skip_segsets        = False # Skipping these would accelerate reading
        model1.read_keyword_file(os.path.join(model_dir,file_name_k), \
            bool_skip_nodes_elements,bool_skip_segsets)
    #
    if True: # Write keyword file
        model1.write_keyword_file(os.path.join(model_dir,"_cask_from_pydyna_.k"))
```

(2/5) Excerpts from the Python script using "PyDyna"

```
    #
    if True: # Define part groups, part tessellations, target masses
        #
        # A "part tessellation" is a set of part groups.
        # It is considered valid:
        # - if its part groups do not share common parts (no overlap),
        # - if it contains all the parts of the model (no hole).
        # This concept is currently unavailable in LS-PrePost.
        #
        # ========================================================================
        #
        pids_body          = [ 101,103,105,356,359,1,2,3,4,5 ]
        pids_content       = [ 106 ]
        pids_primary_lid   = [ 333,334,335,336,337,6 ]
        pids_secondary_lid = [ 338,339,340,341,342,7 ]
        #
        pids_shock_absorber_sheets = [ 344,345,346,347,348,354,357 ]
        pids_shock_absorber_wood_1 = [ 349,350,353 ]
        pids_shock_absorber_wood_2 = [ 351 ]
        pids_shock_absorber_screws = [ 361,362,363,364 ]
        #
        pids_shock_absorber = [  ]
        pids_shock_absorber.extend(pids_shock_absorber_sheets)
        pids_shock_absorber.extend(pids_shock_absorber_wood_1)
        pids_shock_absorber.extend(pids_shock_absorber_wood_2)
        pids_shock_absorber.extend(pids_shock_absorber_screws)
        #
        # (...)
        #
        # ========================================================================
        #
        # Part tessellation for mass distribution check
        #
        ptid = 1 # Part tessellation ID
        target_mass = None
        summed_target_masses = 0.0
        #
        target_mass = 0.5*(5.65e3) # Target mass for primary lid (half model)
        summed_target_masses += target_mass
        pids = pids_primary_lid
        pgid = 1100
        pydyna.C_PartGroup(model1,ptid,pgid,target_mass,"PL etc.",pids).add_to_model()
        #
        # (...)
        #
        target_mass = 0.5*(120.0e3) - summed_target_masses # Target mass of the rest
        summed_target_masses += target_mass
        pids = [ ]
        pids.extend(pids_body)
        pgid = 1600
        pydyna.C_PartGroup(model1,ptid,pgid,target_mass,"ALL",pids).add_to_model()
        #
        model1.check_part_tessellation(ptid) # No overlap? No hole?
        #
        # ========================================================================
```

(3/5) Excerpts from the Python script using "PyDyna"

```
        #
        # Part tessellation for matsum (=> energy absorption per part group)
        #
        ptid = 2
        # (...)
        #
        # ========================================================================
        #
        # Part tessellation for contours (=> stress/strain contours per part group)
        #
        ptid = 3
        # (...)
        #
        # ========================================================================
    #
    if True: # Determine time step and mass scaling per part
        model1.determine_time_step_and_mass_scaling_per_part()
    #
    if True: # Check everything
        model1.check_everything(bool_determine_parts_from_segments = True)
        # If the above Boolean is set to False, "PyDyna" will not provide
        # information about which segment set belongs to which part(s)
        # (the presently implemented algorithm is CPU-intensive).
    #
    #####  POST-PROCESS RESULTS  #####
    #
    if True: # Read ASCII file "messag" (initial timestep, contact stability timesteps)
        model1.read_messag(os.path.join(model_dir,"messag"))
    #
    if True: # Read ASCII file "d3hsp" (check masses)
        options = pydyna.C_Options("read_d3hsp")
        options.part_tessellation_IDs_for_mass_distribution = [ 1 ]
        #
        model1.read_d3hsp (os.path.join(model_dir,"d3hsp" ),options)
    #
    if True: # Read ASCII file "glstat"
        options = pydyna.C_Options("read_glstat")
        options.bool_display_final_energies_on_graph = True
        #
        model1.read_glstat(os.path.join(model_dir,"glstat"),options)
    #
    if True: # Read ASCII file "matsum"
        options = pydyna.C_Options("read_matsum")
        # Internal energies will be summed per part group
        options.part_tessellation_IDs = [ 0, 2 ]
        # Partial sums will be plotted, along with a residual sum less than 0.1%
        options.matsum_cumulated_part_internal_energy_threshold_percentage = 99.9
        options.bool_execute_lspp     = False # Do not plot graphs with LS-PrePost
        options.bool_execute_gnuplot  = True  # Plot graphs with Gnuplot
        #
        model1.read_matsum(os.path.join(model_dir,"matsum"),options)
```

Running “`model1.determine_time_step_and_mass_scaling_per_part()`” prints such lines in the log file:

```
Part          1 :      0.645 kg ;     0.01 m/s ;   … us <= dt_stab <=   … us
  Part name                = <accelerometer>
  Part mass                = 0.645129 kg
  Nodal mass               = 0.645129 kg (deformable area)
  Nodal mass in rigid body = 0.00138262 kg (rigid      area)
  Dependencies:
    <nodal_mass> 0.645129 <- 0.645129 <- {
      'MAT_1_RO': 1.761296199999966e-07,
      'MAT_101_RO': 7.746642780882494e-05}
    <nodal_mass_in_rigbdy> 0.00138262 <- 0.00138262 <- {
      'MAT_1_RO': 1.761296199…e-07}
    <nodal_mass_from_other_parts> 0.645129 <- 0.645129 <- {
      'MAT_1_RO': 1.761296199999966e-07,
      'MAT_101_RO': 7.746642780882494e-05}
```

Mass scaling is thus predicted without running LS-DYNA. Auto-adjustment is made possible.

In general, part masses are linear combinations of material densities and rigid part prescribed masses. Shared nodes between deformable and rigid parts complicate the task of model mass adjustment. “PyDyna” provides the coefficients of these linear combinations, which will make it possible to adjust model masses automatically from linear constraints such as: prescribed or changeable mass densities, part group prescribed masses.

Running “`model1.read_matsum(…)`” creates figures such as Figure 1:

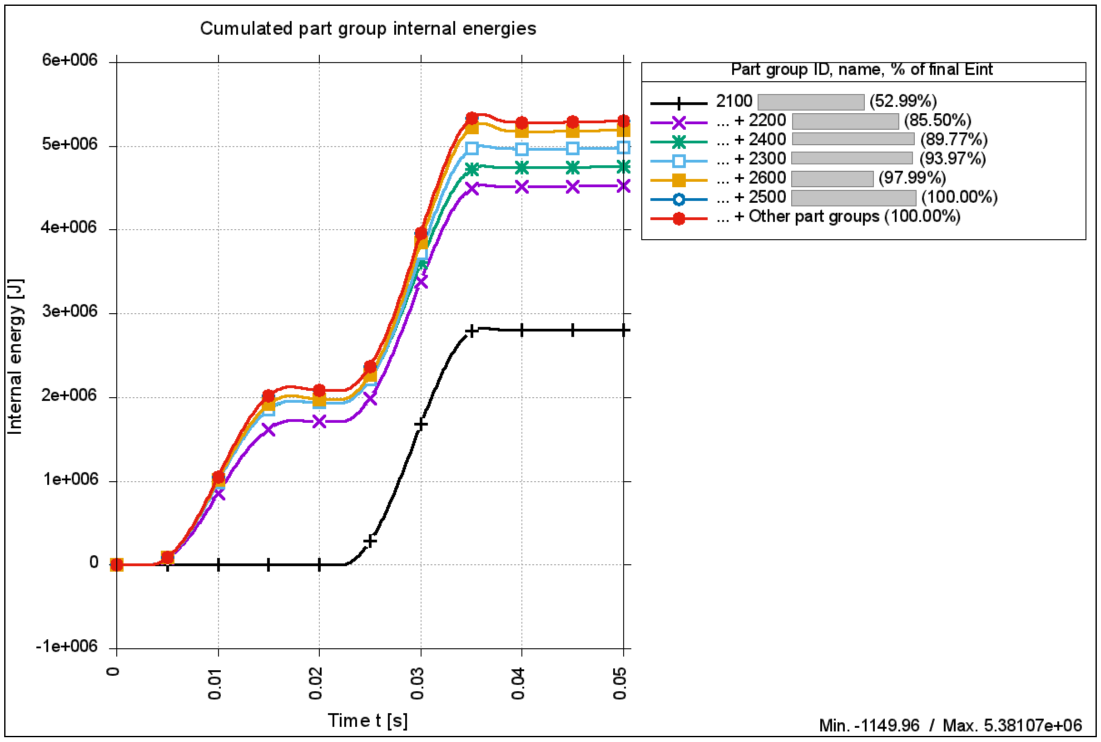

Figure 1: History of “matsum” internal energies, cumulated per part group,
sorted per final value in descending order,
with a residual sum “Other part groups” below 0.1%.

(4/5) Excerpts from the Python script using "PyDyna"

```
    #
    if True: # Read ASCII file "secforc" (screw cross-section forces)
        #
        F0_primary_lid    = XXXe3 # [N]
        F0_secondary_lid  = XXXe3 # [N]
        F0_shock_absorber = XXXe3 # [N]
        #
        prescribed_initial_tension = None
        screw_shank_diameter       = 0.0XX # [m]
        tightening_shear_stress    = XXXe6 # Shear stress [Pa] due to tightening
        yield_stress               = None
        ultimate_stress            = None
        method                     = "BAM-GGR 012 Nov. 2012" # Linearization method
        #
        options = pydyna.C_Options("read_secforc") # Object used to contain options
        #
        options.cross_section_groups = [ ]
        #
        # Append to the list above data about each cross-section group:
        #
        #    options.cross_section_groups.append(
        #         {
        #         "description": "(...)",
        #         # Cross-section IDs:
        #         "csids"      : [ id_offset+k for k in range(1,scr_count+1) ],
        #         # Orientation of x-section axis (for tension/bending components):
        #         "axis"       : "+x",
        #         # The first and last cross-sections belong to half screws:
        #         "sym_planes" : { id_offset+1:"xy", id_offset+scr_count:"xy" },
        #         "prescribed initial tension"     : prescribed_initial_tension,
        #         "screw shank diameter"           : screw_shank_diameter,
        #         "tightening shear stress"        : tightening_shear_stress,
        #         "yield stress"                   : yield_stress,
        #         "ultimate stress"                : ultimate_stress,
        #         "calculate stresses according to": method
        #         } \
        #        )
        #
        # (...)
        #
        # Minima, maxima, averages will be computed in the following ranges:
        # - 1st range [ 1.5 ms , 2.5 ms ] is used to check prescribed initial tension
        # - 2nd range [  49 ms ,  50 ms ] provides residual tension after 9.3m drop
        options.ranges_for_min_max_avg         = [ [  1.5e-3,  2.5e-3 ],
                                                   [ 49.0e-3, 50.0e-3 ] ]
        options.maximum_time                   = None
        options.bool_save_stresses_per_section = True
        options.bool_execute_lspp              = False
        options.bool_execute_gnuplot           = True
        #
        model1.read_secforc(os.path.join(model_dir,"secforc"), options)
```

(5/5) Excerpts from the Python script using “PyDyna”

```
    #
    # (...)
    #
    if True: # Read d3plot files and save stress/strain stats per part to ASCII files
        rd3p_options = pydyna.C_Options("read_d3plot")
        #
        t1 =  0.0e-3
        # Get termination time from LS-DYNA model:
        t2 = model1.get_keyword_field_value("*CONTROL_TERMINATION", None, "ENDTIM")
        dt =  0.1e-3
        # Process “d3plot” files at initial time and final time:
        rd3p_options.only_time_ranges = [ [t1-dt,t1+dt], [t2-dt,t2+dt] ]
        #
        # The effective plastic strain stats will be determined
        # on solid element sets read in an auxiliary keyword file:
        rd3p_options.solid_set_file_name = "solid_element_sets_for_final_eps.k"
        rd3p_options.solid_set_IDs = [ 1,2,3,4,5,6,7,8,9,10,11 ]
        #
        # (...)
        #
        model1.read_d3plot(d3plot_file_name = os.path.join(model_dir,"d3plot"),
                           options = rd3p_options)
    #
    if True: # Plot strain/stress contours
        generalized_part_group_IDs = [ 3100, 3200 ] # Part group IDs are positive
        for pid in [ 362, 312 ]:
            if pid in model1.part_IDs:
                generalized_part_group_IDs.append(-pid) # Part IDs are negative
        generalized_part_group_IDs.extend(
            [ "SolidSet%d" % sid for sid in [ 1,2,3,4,5,6,7,8,9,10,11 ] ])
            # Solid set IDs are given as “SolidSetXXX” strings
        #
        # The reference view is LS-PrePost “top” view (xy-plane),
        # rotated around the graphics window vertical axis by +90°:
        q0 = top().ry(90.0) # q0 is a quaternion; is rotated by function .ry(θ°)
        views = [ ]
        for j in range(0,4+1):
            for i in range(0,4+1):
                # The reference view is rotated around the vertical axis by +/-30° or
                # +/-60°, and then rotated around the horiz. axis by similar angles:
                views.append(q0.ry(30.0*(2-i)).rx(30.0*(2-j)))
        #
        options = pydyna.C_Options("draw_field_contours")
        #
        options.generalized_part_group_IDs = generalized_part_group_IDs
        options.solid_set_file_name        = "solid_element_sets_for_final_eps.k"
        options.views                      = views
        options.title                      = None
        options.bool_identify_extremum     = False
        options.bool_execute_lspp          = False
        options.bool_execute_gnuplot       = True
        #
        model1.draw_field_contours(d3plot_file_name = os.path.join(model_dir,"d3plot"),
                                   options = options)
    #
    # ====================================================================
```

Running “`model1.draw_field_contours(…)`” prints such lines in the log file:

```
   Part group 3100 <Shell> pids={101,103,359}
     Part 101 has EPSU_true = 0.XXXXXX.
     Static range for eps_p : [0.001 ; EPSU_true=0.XXXXX]
```

The part group 3100 named “Shell” contains 3 parts. They all use a *MAT_PIECEWISE_LINEAR_ PLASTICITY material (information obtained in the keyword file). This material models the behavior of metals. The static range for the effective plastic strain contours is then automatically set to [0.1% ; ultimate effective plastic strain ε_u of the first material]. No warning message says that the other 2 materials have different values of ε_u. Should the case have arisen, the upper bound would have been set to the minimum of the 3 values.

Conclusion

ORANO TN makes continuous effort to improve LS-DYNA analyses when evaluating the crashworthiness of casks for the transportation or storage of radioactive material.

Part of the effort is dedicated to Verification & Validation.

Considerable effort is also being made to improve Productivity and Quality. For more than two years, a Python library has been developed as a comprehensive tool that covers most of the phases of the analysis process, with fruitful usage of model data in the post-processing phase.

References

[1] International Atomic Energy Agency (IAEA): “Regulations for the Safe Transport of Radioactive Material – Safety Requirements – IAEA Safety Standards Series No. TS-R-1”, 2009 Edition, Vienna, Austria, 2009

[2] LIVERMORE SOFTWARE TECHNOLOGY CORPORATION (LSTC): “LS-DYNA Keyword User's Manual – Volumes I and II – Version 971 R6.1.0”, August 2012.

[3] Marchaud G., Saint-Jean V. (AREVA TN): “Verification and Validation of LS-DYNA for the Transport and Storage of Radioactive Materials”, Proc. 11th Eur. LS-DYNA Conf., Salzburg, Austria, 2017.

[4] LS-PrePost web site www.lstc.com/lspp/ [Accessed Jan. 11, 2018].

[5] Perl programming language web site www.perl.org [Accessed Jan. 11, 2018].

[6] Python programming language web site www.python.org [Accessed Jan. 11, 2018].

[7] DYNAmore web site, LS-DYNA-Tools page www.dynamore.de/en/products/tools [Accessed Jan. 15, 2018].