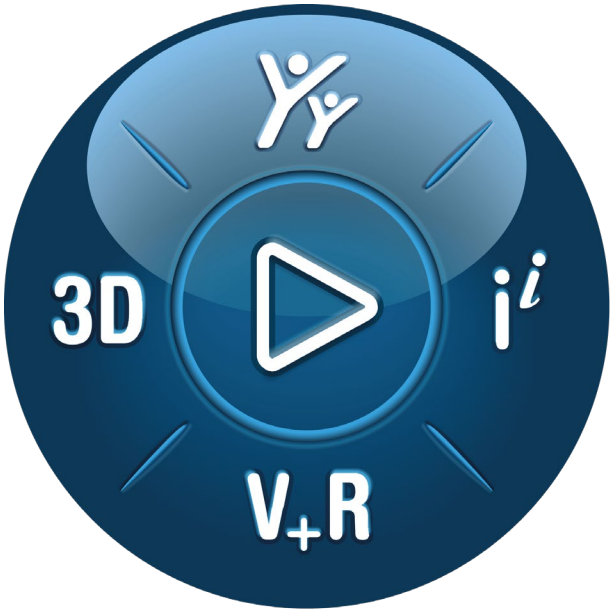
# Simulation Using the Iterative Solver Technology in Abaqus




3DEXPERIENCE®



Version 1.0 - 4/14/2022

Written by: Tao QU
Validated by: Vladimir BELSKY,
Mintae KIM, David PALMER
Edited by: Arati DESAI

**SIMULIA 3DEXPERIENCE R2022x FP2205**

DASSAULT SYSTEMES | The 3DEXPERIENCE® Company

# Executive Summary

***This document describes best practices for a new, state-of-the-art iterative linear equation solver in Abaqus/Standard and the 3DEXPERIENCE structural simulation apps.***

***The new solution capability is based on an original, unpublished proprietary algorithm developed for reliability and efficiency. A scalable parallel implementation results in a very fast, low memory consumption solver well suited for very large models.***

***Together with the high performance modeling, meshing, and visualization capabilities of the 3DEXPERIENCE platform, this technology empowers structural and mechanical analysts with a new tool to handle large, real-world simulation models with over 100 million degrees of freedom.***

***The iterative solver is best suited to models with certain characteristics. This Best Practices document provides usage guidelines and strategies to help you best use this technology.***

***This document is applicable to all releases from 3DEXPERIENCE R2022x and above.***

***Prerequisites and recommendations: To take full advantage of this document, you must be familiar with the SIMULIA product Abaqus.***

***Target audience: Structural and Mechanical Analysts that develop large, computationally intensive finite element models.***

# Table of contents

# 1. About the Iterative Solver

The solution of systems of linear equations is an integral part of finite element analysis. A nonlinear analysis in Abaqus/Standard is solved in an incremental and iterative approach using the Newton method (or its variants). In this approach it is necessary to solve a system of linear equations in each iteration. Two equation solver types are available in Abaqus/Standard – direct and iterative.

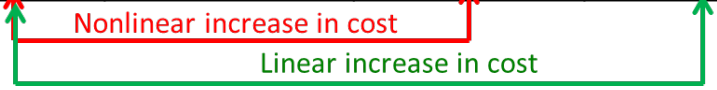
A nonlinear analysis may require many increments, and subsequently many passes through the equation solver. The performance of the equation solver becomes more critical as the model size and complexity grows.

The computational cost of an equation solution can be described in terms of memory use and floating point operations (FLOPs) per solver pass. The FLOP count of the direct solution grows quickly as the model size increases. This is exacerbated for models that exhibit a high degree of connectivity and a uniform aspect ratio in all three dimensions (that is, cube-like geometry). For example, when analyzing a 1 million degree of freedom (DOF) cube model, the number of FLOPs per solver pass is in the order of 1.0E+13. Whereas, for a 10 million DOF cube it approaches 1.0E+15.

Computation exercises have demonstrated that problems of this nature are well suited for iterative solution methods. This is because the computational cost often scales more linearly with DOF count than the direct methods. Costs of the two approaches are compared in Table 1.1 for a representative model, where the FLOP count is given for the direct solver.

*Table 1.1: Comparison of computational cost of the Abaqus/Standard direct and iterative solvers as model size increases*

| | DoFs (Millions) | Factorization FLOPs | Direct Solver Wallclock Time (hrs.) | Iterative Solver Wallclock Time (hrs.) |
|---|---|---|---|---|
| Problem 1 | 7.1 | 5.6E+13 | 0.11 | 0.02 |
| Problem 2 | 32.0 | 1.4E+15 | 3.30 | 0.11 |
| Ratio (Problem2 / Problem1) | 4.5 | 25 | 30 | 5.5 |




The iterative linear equation solver in Abaqus/Standard is most useful for “bulky” models involving a large number of variables (typically counted in millions of DOFs). For models of this class, the expense of the direct solver can become prohibitive. In such situations the advantages of the iterative solver are:

- Performance
  - dramatically faster than the direct solver, and
  - significantly less memory usage than the direct solver

- Scalability
  - solution cost grows approximately in proportion to the size of the model, and
  - hybrid parallelization approach based on both MPI- (Message Passing Interface) and threads allows optimal use of compute resources

## 1.1. Choosing the Iterative Solver

Iterative solution techniques differ from direct techniques in that their successful use is somewhat problem-dependent. The computational performance and convergence behavior of an iterative solver is maximized for certain classes of problems; specifically, those with high mesh connectivity and low sparsity.

The iterative linear equation solver in Abaqus/Standard can be used for linear and nonlinear static, quasi-static, heat transfer, geostatic, and coupled pore fluid diffusion and stress analysis procedures.

The iterative solver technology implemented in Abaqus/Standard allows you to include complex material nonlinearities, constraints, and contact. However, there are several factors that you should consider when deciding on a solution technique.

### 1.1.1. Model Size

The iterative solver should be used for large models that would otherwise require a prohibitively large number of FLOPs by the direct solver. Figures 1.1 and 1.2 offer some comparison of the iterative and direct solvers for a group of representative industrial models. In both graphs, the FLOP count per direct solver pass is presented on the left vertical axis; specifically, the maximum FLOP count recorded during the solution process.

On the respective right vertical axes, Figures 1.1 and 1.2 show the wall clock speedup factor and the memory reduction obtained by running the analyses using the iterative solver.

Each model in the plot is characterized with two measures: DOFs and FLOPs. For example, the model with 25 million DOFs has a 1.0E+15 max FLOP count, and using the iterative solver for this model shows a performance speedup factor of 3.78. In addition, there is a 52% memory reduction as compared with the direct solver.

In general, for models greater than 10 million DOFs and requiring more than 1.0E+12 FLOPs per solver pass, the iterative solver is expected to outperform the direct solver.

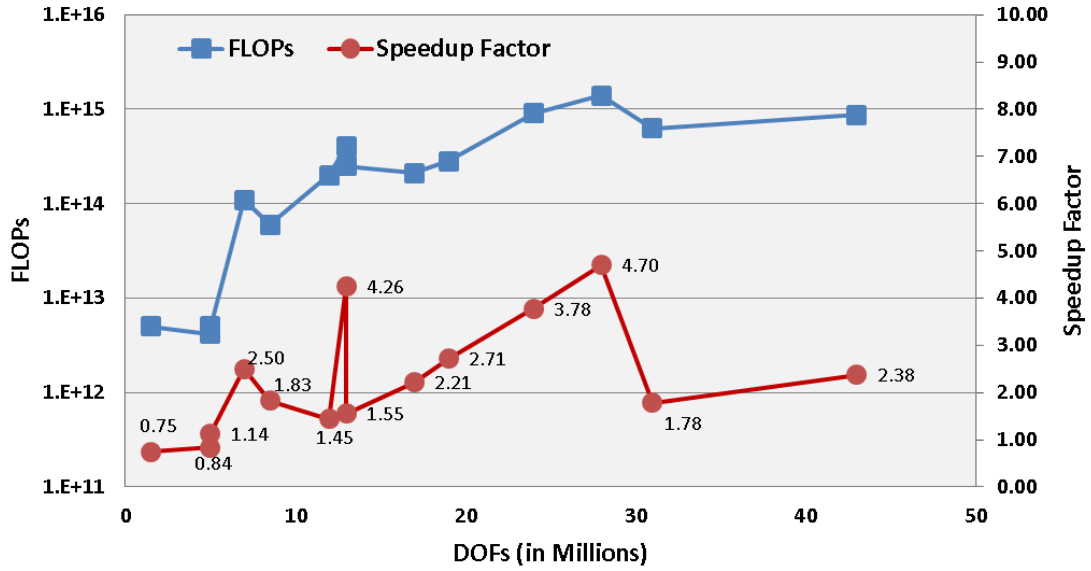



*Figure 1.1: Iterative solver speedup over the direct solver for industrial benchmark models. For example, the model with 25 million DOFs requires 1.0E+15 FLOPs and the iterative solver shows a 3.78 speedup factor over the direct solver.*

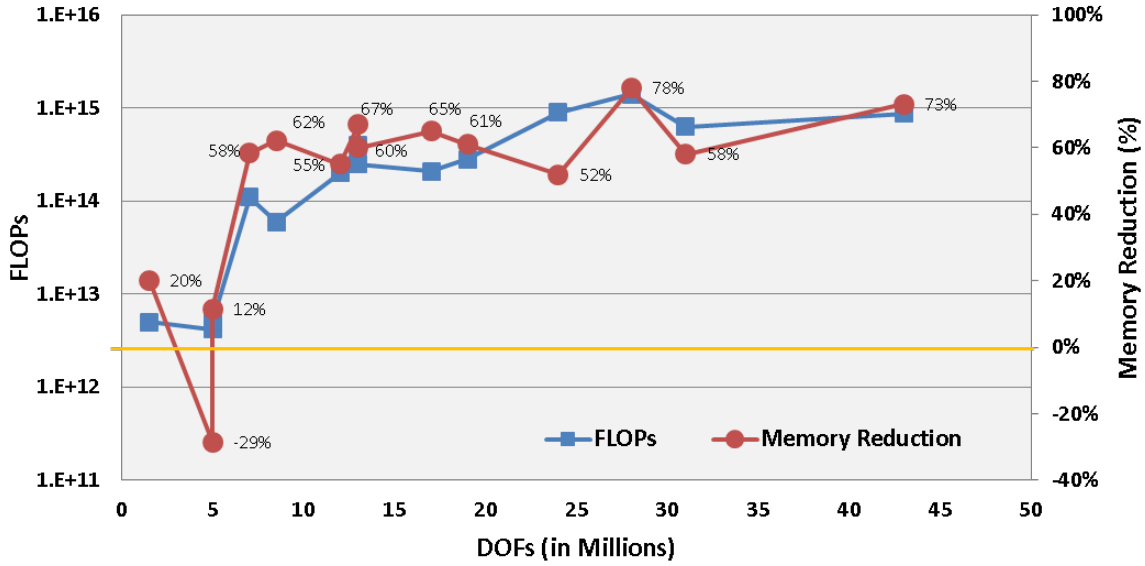



*Figure 1.2: Iterative solver memory reduction over the direct solver for industrial benchmark models. For example, the model with 25 million DOFs requires 1.0E+15 FLOPs and the iterative solver reduces the memory consumption by 52% over the direct solver.*

### 1.1.2. Geometry and Assembly of the Model

The performance of the iterative solver relative to the direct sparse solver depends on the model geometry and complexity. Structures with a high degree of mesh connectivity and a relatively low degree of sparsity are better suited for an iterative solution. Specifically

- Blocky type structures, individually or in assemblies with a small number of parts, such as the typical powertrain model shown in Figure 1.3, are well suited for the iterative solver.

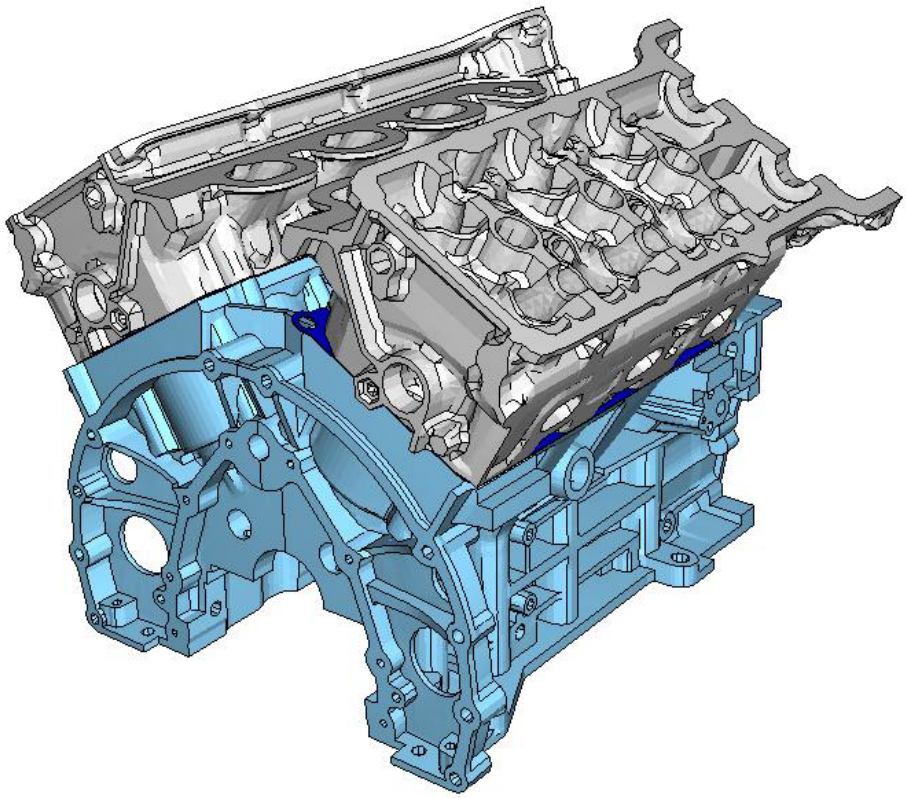


*Figure 1.3: A typical powertrain model. Such models often contain a number of bolts with pre-tension loads that connect the head to the engine block. Contact is defined between the gasket and the head and between the gasket and the engine block.*

Models of assemblies with a large number of repetitive components are not suited for the iterative solver. See Figure 1.4 and 1.5 for such examples. Figure 1.4 shows an industrial benchmark model consisting of 300 stacked layers (each layer is meshed with solid elements) with contact pairs. Figure 1.5 shows an industrial benchmark model with 9000 parts (each layer is meshed with solid elements) interacting through general contact.

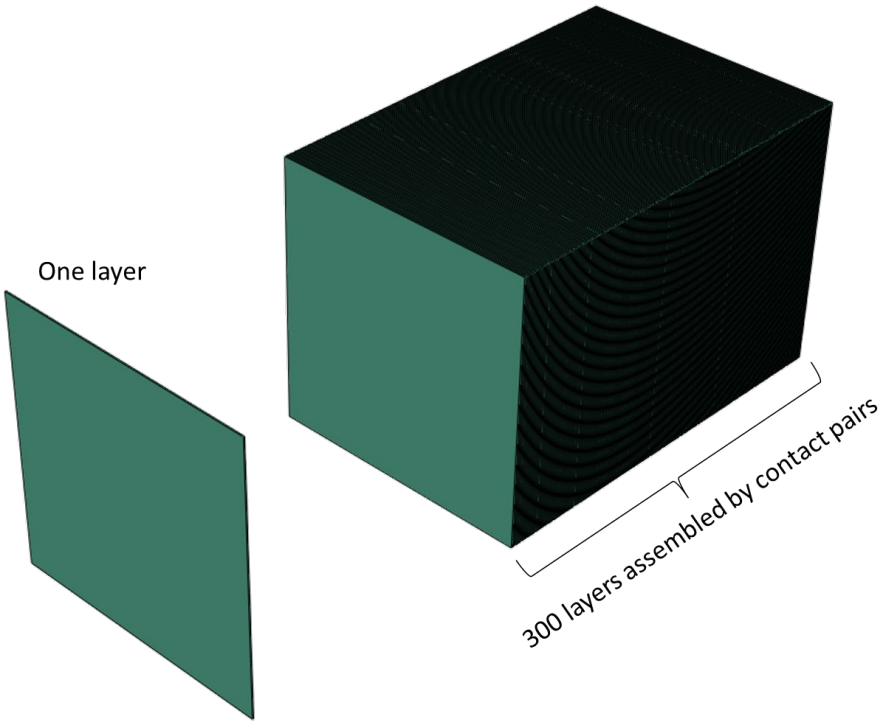



*Figure 1.4: Benchmark model of 300 stacked solid layers with corresponding contact pairs*

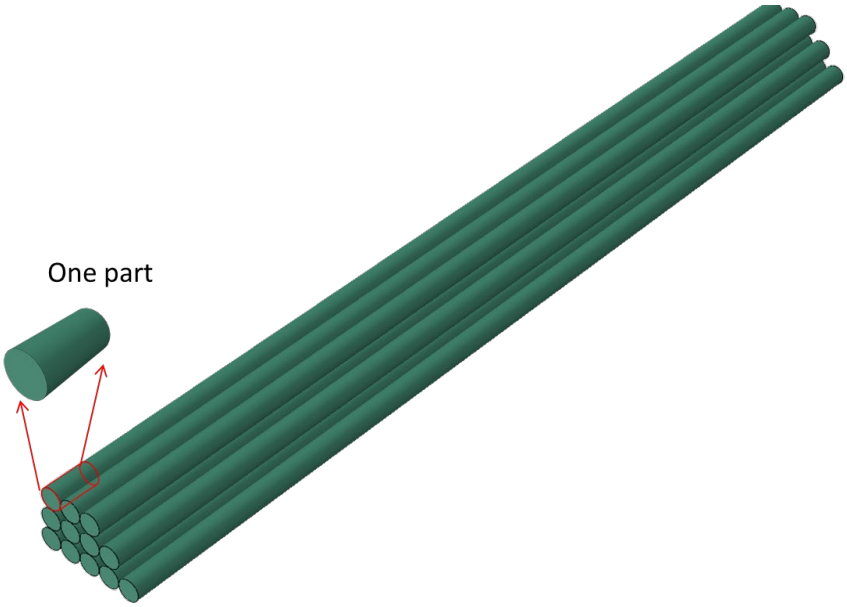



*Figure 1.5: Benchmark model consisting of thousands of individual solid parts interacting through general contact*

Other factors that can impact the robustness and performance of the iterative solver include element types, constraint equations, material and geometric nonlinearities, and changes in contact interactions. For additional information, see Deciding to use the iterative solver.

# 2. Accessing the Iterative Solver

On the 3DEXPERIENCE platform, you can access the iterative solver in the SIMULIA Structural/Mechanical Scenario Creation apps. You can use the Structural Analysis Engineer (DRD) or Mechanical Analysis Engineer (SMU) roles (Figure 2.1).

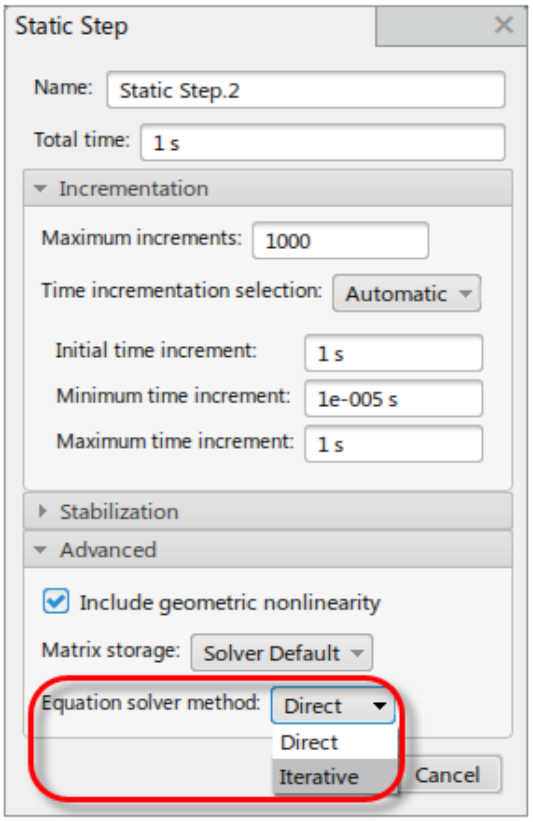



*Figure 2.1: Access to the iterative solver on 3DEXPERIENCE*

With an Abaqus input file, you can use the following keywords to invoke the iterative solver in each step definition:

```
*STEP, SOLVER=ITERATIVE
```

In Abaqus/CAE, access the iterative solver in the Step module (Figure 2.2) for each step:

**Step module: Step editor: Other: Method: Iterative**

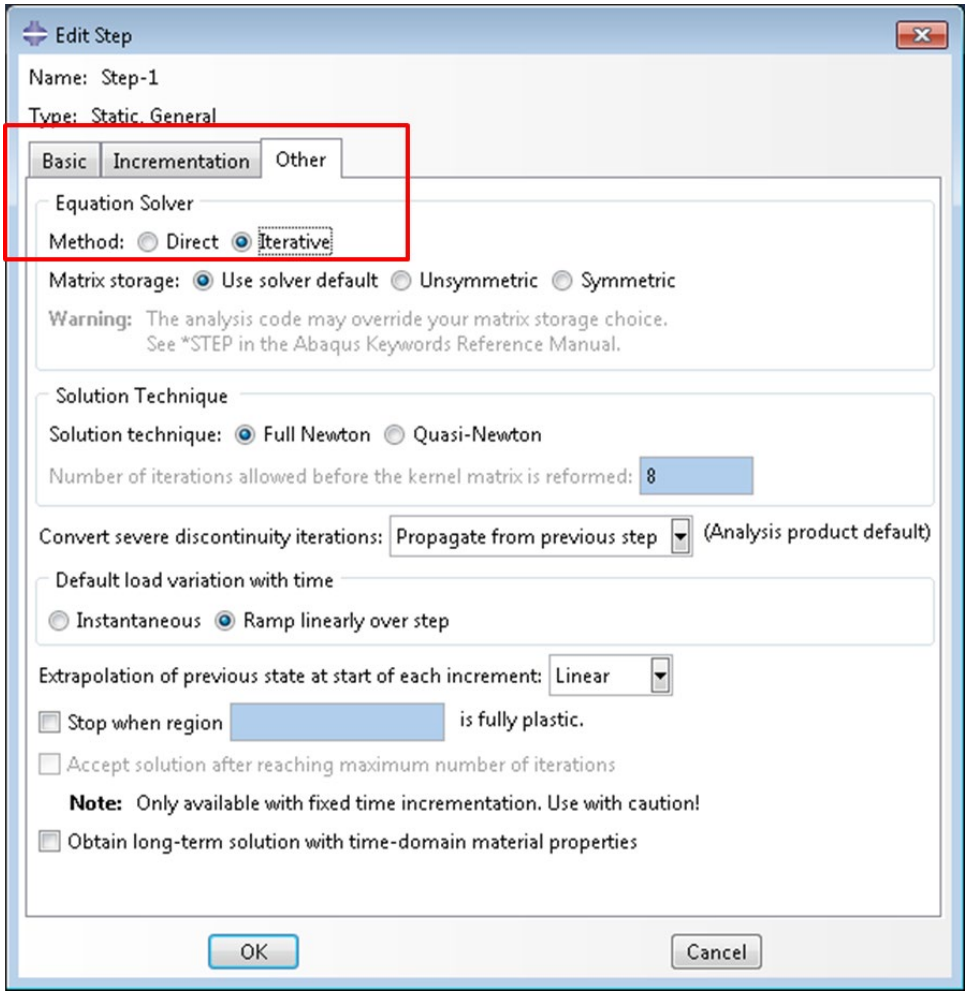



*Figure 2.2: Access to the iterative solver in Abaqus/CAE*

If an iterative solver is requested in the Step definition for geostatic (*GEOSTATIC) or coupled pore fluid diffusion and stress (*SOILS) analyses, a specialized single level iterative solver is currently activated by default. If an algebraic multi-level iterative solver is required for these analyses (which is not currently recommended), you require to include the following environment variables in the “abaqus_v6.env” file:

```
os.environ[‘ABA_ITERATIVE_SOLVER_NEW’]='4'
mp_environment_export=mp_environment_export+('ABA_ITERATIVE_SOLVER_NEW',)
```

This setting was implemented in 2019x FD02 (FP 1914).

In addition, when running analyses with the iterative solver, consider including the following environment variables:

- Increase the memory to 90% of the physical memory per host (note: the default memory policy setting is 80%).

  ```
  memory = "90%"
  ```

- Invoke parallel execution in preprocessing all the time to avoid size limits of the global database structure for large models for restart analyses (*RESTART, WRITE).

  os.environ['ABA_PRE_DECOMPOSITION']='1'

  mp_environment_export=mp_environment_export+('ABA_PRE_DECOMPOSITION',)

- Generate use (.use) files to aid in monitoring run time and memory usage.

  os.environ['ABA_RESOURCE_MONITOR']='ON'

  mp_environment_export=mp_environment_export+('ABA_RESOURCE_MONITOR',)

# 3. Setting Controls for the Iterative Solver

The Abaqus/Standard iterative solver has a number of numerical controls; these are unique to the iterative solution technique and are separate from the controls used by the Newton method. Additional discussion about the overall convergence of an analysis is offered in Section 5.2.

## 3.1. Convergence Tolerance and Maximum Number of Iterations

By default, the convergence tolerance of the iterative linear equation solver in Abaqus/Standard is 1.0E-06, which typically is sufficient to maintain the convergence of the Newton method. The default number of iterations allowed for the iterative solver (not to be confused with the maximum number of nonlinear Newton iterations) is 500 per solver pass. In rare instances, it may be necessary to tighten the tolerance (for example, to 1.0E-08) and/or increase the maximum number of iterations (for example, to 1000).

To specify the convergence tolerance of the iterative solver, use the following keywords in the input file:

*SOLVER CONTROLS

<tolerance>

Or, the following procedure in Abaqus/CAE (Figure 2.3):

**Step module: Other > Solver Controls > Edit: Specify: Relative tolerance: Specify: <tolerance>**

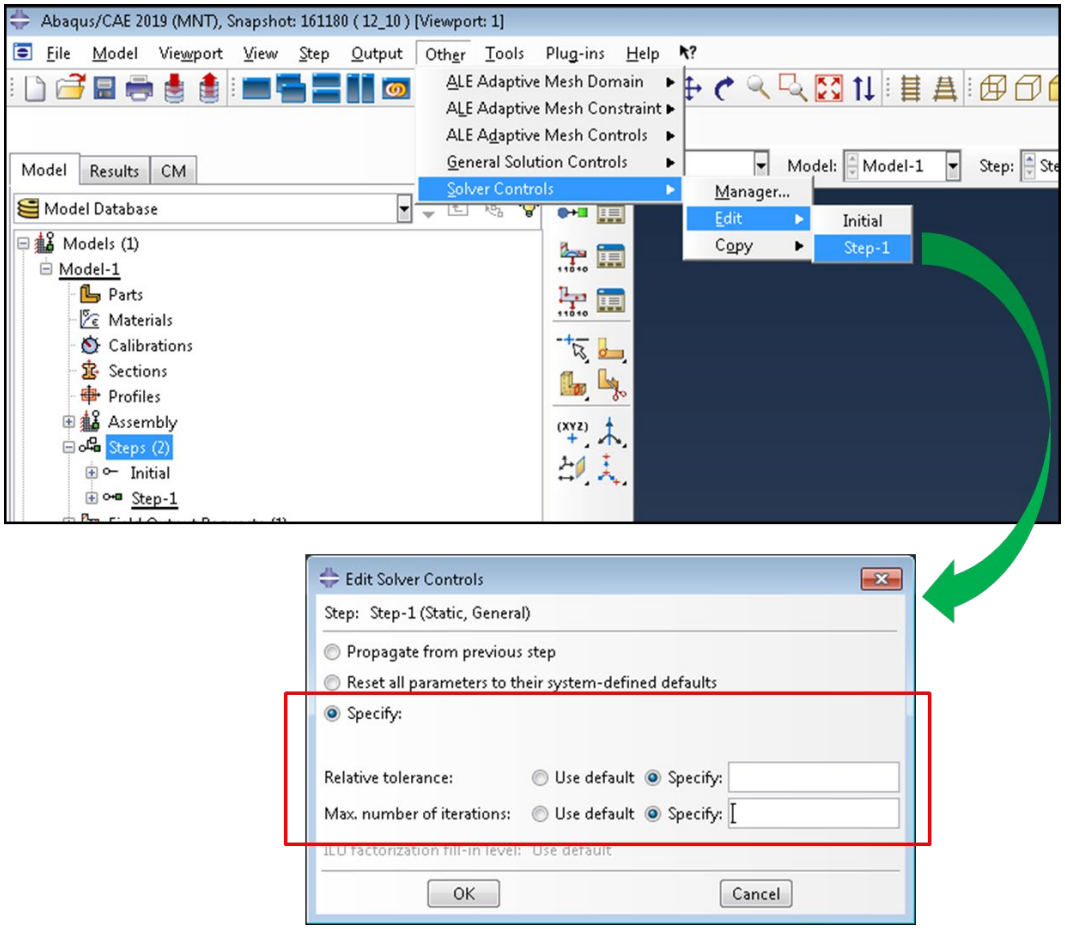



*Figure 2.3: Setting the solver controls in Abaqus/CAE*

To specify the maximum number of iterations, use the following keywords in the input file:

```
*SOLVER CONTROLS
, <maximum number of iterations>
```

Or, the following procedure in Abaqus/CAE (Figure 2.3):

**Step module: Other > Solver Controls > Edit: Specify: Max. number of iterations: Specify: <maximum number of iterations>**

In the input file or Abaqus/CAE, specify the solver control setting for each step. To change the default values for all steps in the entire analysis, the following environment variables, which take precedence over the keywords entry, can be used for specifying the tolerance:

```
os.environ['ABA_EQSITR_RTOL'] = '<tolerance>'
mp_environment_export=mp_environment_export+('ABA_EQSITR_RTOL',)
```

And, for specifying the maximum number of iterations:

```
os.environ['ABA_EQSITR_MAX_ITER'] = '<maximum number of iterations>'
mp_environment_export=mp_environment_export+('ABA_EQSITR_MAX_ITER',)
```

### 3.2. Smoother Types

The Abaqus/Standard iterative solver uses a proprietary algebraic multigrid (AMG) implementation. Coarsening and smoothing is fundamental to any AMG approach, and the smoother is a key factor that can affect the performance of the iterative solver. The iterative solution technique in Abaqus/Standard offers three types of smoothers for both symmetric and unsymmetric problems; environment variables are used to select the type.

The default smoother type for symmetric problems is “weak_sparse” which typically provides the optimal solver configuration for large models with complex features:

```
os.environ['ABA_PAMG_SMOOTHER_TYPE'] = 'weak_sparse'
mp_environment_export=mp_environment_export+('ABA_PAMG_SMOOTHER_TYPE',)
```

For unsymmetric problems, the default smoother type is “unsym_weak_sparse”.

The “jacobi_chebyshev” smoother is suited for models with simple modeling features, for example, a bulky model without contact, constraints (such as *MPC, *EQUATION, *COUPLING, etc.), gasket elements, or pre-tension sections. It may provide faster performance but could potentially be less robust:

```
os.environ['ABA_PAMG_SMOOTHER_TYPE'] = 'jacobi_chebyshev'
mp_environment_export=mp_environment_export+('ABA_PAMG_SMOOTHER_TYPE',)
```

This smoother type can be used for both symmetric and unsymmetric problems.

For models that exhibit convergence difficulties when using the “weak_sparse” and “jacobi_chebyshev” smoothers, it may be necessary to try “strong_sparse”. It is typically the most robust, but at the expense of performance:

```
os.environ['ABA_PAMG_SMOOTHER_TYPE'] = 'strong_sparse'
mp_environment_export=mp_environment_export+('ABA_PAMG_SMOOTHER_TYPE',)
```

The corresponding unsymmetric smoother type is “unsym_strong_sparse”.

## 4. Parallel Execution Using the Iterative Solver

The Abaqus/Standard iterative solver is a powerful tool for the analysis of very large models. As such, it will normally be executed in parallel. In this section we discuss the details of multi-core execution.

## 4.1. Basics of SMP, DMP and Hybrid Configuration

The parallelization scheme of the iterative solver follows that of the Abaqus/Standard direct solver by using a hybrid MPI- and thread-based approach. By default, Abaqus/Standard uses only thread-based parallelization when running on a single host and hybrid parallelization (both MPI and threads) when running on multiple hosts. The default number of threads per MPI process is equal to the number of processor cores assigned to each host.

For example, when a job is submitted to a single host of 16 cores using the following command:

**abaqus -job JobName -cpus 16**

by default, the job runs with one MPI process of 16 threads. This is a Shared Memory Parallelization (SMP) configuration, as shown in Figure 2.4.

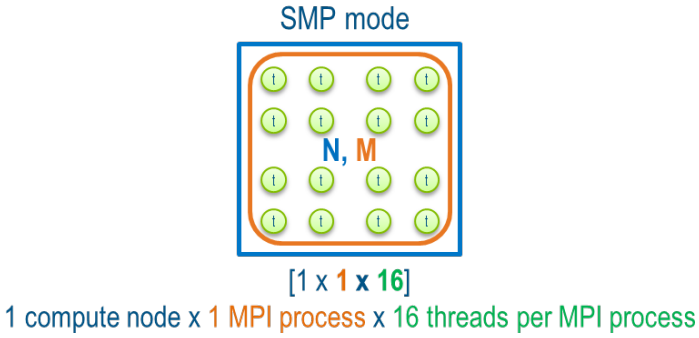



*Figure 2.4: Schematic diagram of the SMP configuration*

The “threads” parameter can be used to limit the number of threads used per MPI process. The value of this parameter is used with the number of cores requested to determine the number of MPI processes. When submitting a job using the following command:

**abaqus -job JobName -cpus 16 -threads 1**

the job runs with 16 MPI processes with 1 thread per MPI process. This is a pure Distributed Memory Parallelization (DMP) configuration, as shown in Figure 2.5.

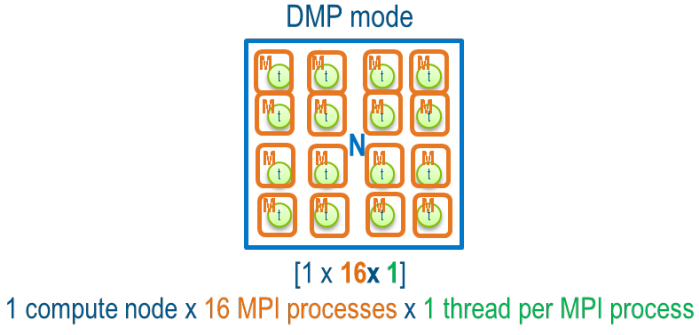



*Figure 2.5: Schematic diagram of the pure DMP configuration*

A hybrid MPI- and thread-based parallelization configuration is achieved when submitting the job, for example, using the following command:

**abaqus -job JobName -cpus 16 -threads 4**

The job runs with 4 MPI processes with 4 threads per process, as shown in Figure 2.6. The hybrid configuration allows you to balance performance with available memory resources.

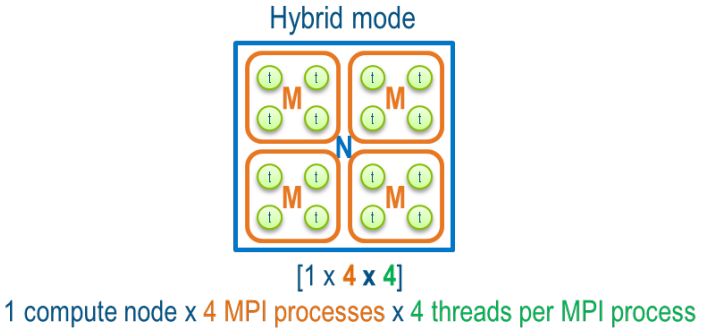



*Figure 2.6: Schematic diagram of the hybrid configuration*

For additional information on SMP, DMP and hybrid configurations, see DS Knowledge Base articles Using MPI-based parallelization with the Abaqus/Standard direct solver and Performance and memory considerations when using the Abaqus/Standard AMG iterative solver.

## 4.2. Optimal Settings for Running the Iterative Solver

The performance discussed in this section is measured only by the wallclock time for Abaqus/Standard, reported as the value of “WALLCLOCK TIME” in the message (.msg) file. The maximum memory consumption of Abaqus/Standard is read as the value of column “MaxRSS” in the line marked with “std-End” in the use (.use) file.

The best performance of the iterative solver is achieved when using a pure DMP configuration (one thread per MPI process). However, the overall memory consumption for Abaqus/Standard may increase significantly because the model must be decomposed into a large number of domains. An analysis using the iterative solver cannot run if the memory requirement of the analysis exceeds the memory policy setting (by default, 80% of the physical memory per host).

If memory is at a premium, the hybrid mode consumes less memory than pure DMP mode. In this case, try using 4 threads per MPI process (threads = 4). The memory footprint of the iterative solver decreases by reducing the number of MPI processes (the number of domains) and employing multiple threads inside each MPI process.

To demonstrate the effect of the “threads” parameter on the performance, benchmark tests were performed with different threads options using the iterative solver. For more information, see DS Knowledge Base article Performance and memory considerations when using the Abaqus/Standard AMG iterative solver.

# 5. Diagnostics

A nonlinear analysis in Abaqus/Standard often has challenging and complex physics. Not all modeling features are amenable to iterative solver analysis. In this section we offer guidance on resolving potential convergence difficulties and activating diagnostic output for the iterative solver.

## 5.1. Current Limitations

There are a few cases where the iterative solver cannot be used with the current implementation (subject to change in future releases).

- Geostatic (*GEOSTATIC) or coupled pore fluid diffusion and stress (*SOILS) analyses for the models with Lagrange multipliers are not supported, and the following error message is issued by the preprocessor in the data (.dat) file:

  ***ERROR: ITERATIVE SOLVER CAN NOT BE USED FOR MODELS THAT CONTAIN LAGRANGE MULTIPLIERS IN THE GEOSTATIC OR SOIL ANALYSES. PLEASE REMOVE FEATURES THAT REQUIRE THE USE OF LAGRANGE MULTIPLIERS (E.G. DISTRIBUTING COUPLINGS, HYBRID ELEMENTS, CONNECTOR ELEMENTS, HARD CONTACT) OR USE THE DIRECT SOLVER.

- Analyses with automatic stabilization with an adaptive damping factor are not supported, and the following error message is issued by the preprocessor in the data (.dat) file:

  ERROR: THE ITERATIVE EQUATION SOLVER CANNOT BE USED WITH STABILIZATION PARAMETER WITH ADAPTIVE AUTO DAMPING FOR *STATIC PROCEDURE. INCLUDE ALLSDTOL=0.0 TO TURN OFF ADAPTIVE AUTO DAMPING.

  As a workaround, a constant damping factor can be used if numerical stabilization is necessary. For additional information, see Automatic stabilization of unstable problems.

- Analyses that include both the iterative solver and an eigensolver (see Natural frequency extraction) are not supported, and the following error message is issued in the preprocessor in the data (.dat) file:

  ERROR: ITERATIVE SOLVER CAN NOT BE USED WITH *FREQUENCY PROCEDURE.

  Instead, the direct solver should be used.

- Analyses with *INERTIA RELIEF are not supported by default, and the following error message is issued in the preprocessor in the data (.dat) file:

  ERROR: INERTIA RELIEF LOADS MAY NOT BE USED WITH THE ITERATIVE SOLVER.

However, the iterative solver can be activated for the analysis with *Inertia Relief through the following environment variable if a model has statically determined boundary conditions or is properly constrained:

os.environ[‘ABA_EQSITR_INERTIA_RELIEF’] = 'ON'

mp_environment_export += ([‘ABA_EQSITR_INERTIA_RELIEF’,)

## 5.2. Resolving Convergence Difficulties

In a nonlinear analysis, Abaqus/Standard breaks the simulation step into a number of time increments and finds the approximate equilibrium configuration at the end of each time increment. In a time increment, Abaqus/Standard uses the Newton method, possibly taking several iterations, to determine an acceptable solution.

### 5.2.1. Convergence of the Iterative Solver in Abaqus/Standard

In a Newton iteration, Abaqus/Standard uses the structure’s tangent stiffness, K, and the force residual, f, to calculate a displacement correction u. This is done by solving a system of linear equations (Ku = f) using the iterative solver (not to be confused with the nonlinear Newton iteration stated above). The error of the approximate solution is measured by the relative residual of the linear system of equations, defined by || Ku – f || / || f ||. The convergence of the iterative solver is achieved when this relative residual is below a specified tolerance (by default, the tolerance is 1.0E-06 as discussed in Section 2.2.1).

If the relative residual falls below the tolerance within a specified number of iterations (by default, the maximum number is 500 as discussed in Section 2.2.1), the iterative solver is considered to have achieved a converged solution. Otherwise, Abaqus/Standard issues a warning message in the message (.msg) file: ”WARNING: NON-CONVERGED SOLUTION FROM THE ITERATIVE SOLVER.” Note that even in the presence of this warning, the displacement correction computed at the end of the maximum number of iterations is used to check the convergence of the Newton iteration.

### 5.2.2. Convergence of the Newton Method in Abaqus/Standard

Upon calculation of the displacement correction, the structure’s new configuration is determined, and a new force residual is calculated using the internal forces in the new configuration. At an iteration of the Newton method, if both the force residual and the displacement correction are less than their respective tolerance values (for more details, see Solving nonlinear problems), the solution is said to have converged for the current time increment. If the convergence criteria are not satisfied, Abaqus/Standard issues messages in the message (.msg) file, such as “FORCE EQUILIBRIUM NOT ACHIEVED WITHIN TOLERANCE”, or “DISP. CORRECTION TOO LARGE COMPARED TO DISP. INCREMENT.” If necessary, Abaqus/Standard performs further Newton iterations.

It is important to note that the convergence tolerance of the iterative linear equation solver is independent of the convergence tolerances of the nonlinear Newton iterations. The latter are

used to determine whether a solution increment converges and are the same regardless of the choice of linear equation solver, iterative or direct.

### 5.2.3. Convergence Difficulties in the Iterative Solver

It is possible that the iterative solver encounters convergence difficulties while the Newton process continues to converge, even with a non-converged solution obtained from the linear equation solver. If this pattern occurs quite frequently, it suggests that the default maximum number of iterations (that is, 500) is not sufficient to effectively reduce the relative residual to the desired tolerance. In this case, increase the maximum number of iterations (for example, to 1000) and/or change the smoother type (as discussed in Section 2.2.2). For problems with mild nonlinearities, it may also be acceptable to loosen the convergence tolerance of the iterative solver (for example, to 1.0E-03).

### 5.2.4. Convergence Difficulties of the Newton Process

It is also possible that the nonlinear Newton process fails to converge while the iterative solver converges. Very rarely, this non-convergence of the Newton process is due to the approximate solution obtained by the iterative solver; in this case it may be necessary to improve the accuracy of the solution by tightening the convergence tolerance of the iterative solver (for example, to 1.0E-08). In most cases, non-convergence of the Newton process is due to modeling issues. In complex models, the Newton process may lose convergence for a number of reasons; the following sections briefly discuss some common issues and possible remedies.

- Highly distorted elements

  Look for diagnostic messages related to the mesh quality in the data (.dat) file. Elements with very large aspect ratios (as shown in Figure 2.7) can lead to convergence difficulties and in many cases cause the iterative solver to fail to converge. When the direct solver is used instead, the convergence can be improved but the results are likely to be inaccurate in the area where poor quality elements are present.

  In this case, the mesh quality should be improved for both better convergence and better solution accuracy. Highly distorted elements are included in the element set named “WarnElemDistorted” written in the output database (.odb) file, which can be visualized using 3DEXPERIENCE Physics Results Explorer or Abaqus/Viewer. Sometimes nodal position adjustments made by tie constraints and contact interactions can deteriorate element quality as well. Run a datacheck analysis and confirm that node adjustments have not introduced element distortion. Note that the element quality checks are performed after such adjustments are made.

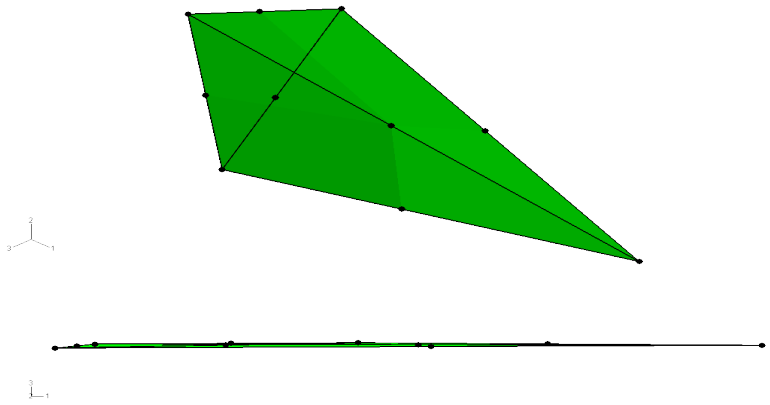


*Figure 2.7: Isometric view (upper) and side view (lower) of a C3D10 element with nearly zero volume. The element is essentially planar as seen in the lower image. This element has a poor volume quality measure of 1.0E-26.*

- Contact interactions

  If contact is involved, use the penalty method (*SURFACE BEHAVIOR, PENALTY) or the augmented Lagrange method (*SURFACE BEHAVIOR, AUGMENTED LAGRANGE) instead of the direct method. Scale down the penalty stiffness to overcome convergence difficulties if necessary, making sure that the resulting penetrations are not excessive. For more details, see Contact constraint enforcement methods in Abaqus/Standard.

  If second-order tetrahedral elements, such as C3D10, are involved in the contact interaction, use the surface-to-surface contact formulation (*CONTACT PAIR, TYPE=SURFACE TO SURFACE) instead of the node-to-surface contact formulation (see Three-dimensional surfaces with second-order faces and a node-to-surface formulation). It is also important to understand how Abaqus/Standard interprets and resolves the contact conditions at the beginning of a step or an analysis. If necessary, check initial contact conditions in the message file (see Difficulties resolving initial contact conditions and The Abaqus/Standard message file). Unintended contact openings or overclosures can lead to poor interpretations of surface geometry, unintended motion in a model, and even non-convergence of an analysis.

- Gasket elements

  Many powertrain models contain gaskets between the cylinder head and the engine block and between the oil pan and the engine block. For gaskets in contact with continuum elements, the small-sliding, surface-to-surface contact formulation (*CONTACT PAIR, SMALL SLIDING, TYPE=SURFACE TO SURFACE) is recommended. For more details, see Small-sliding, surface-to-surface contact. If the gasket elements are unsupported by the surrounding structure through tie constraints or contact (as shown in Figure 2.8), stabilization is required. Specify a

small artificial stiffness for the gasket section by the keywords *GASKET SECTION, STABILIZATION STIFFNESS=<stiffness value>, see Stability of unsupported gasket elements.

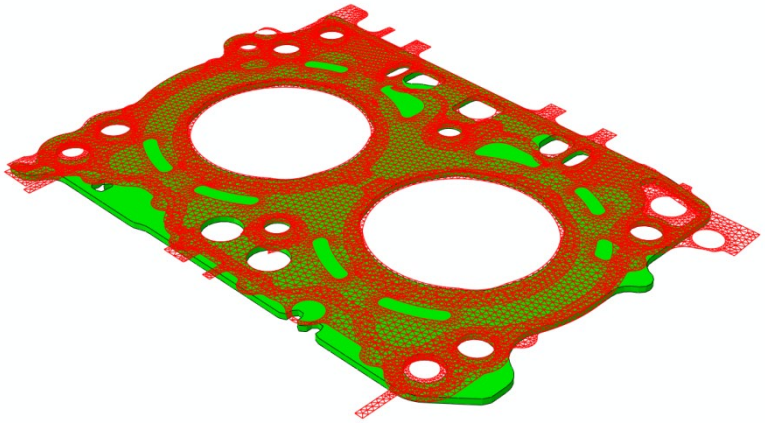


*Figure 2.8: Schematic showing a gasket (green) with areas unsupported by neighboring components in contact (red)*

For problems with persistent non-convergence, you may want to run the analysis using the direct solver. The modeling setup can be tested to determine whether the non-convergence is due to the usage of the iterative solver. Examine the warning messages in the message (.msg) file for numerical singularities, zero pivots and negative eigenvalues, and resolve these issues. For more details, see DS Knowledge Base articles What do ZERO PIVOT warnings in the message file of my Abaqus/Standard analysis mean and Understanding Abaqus/Standard negative eigenvalue messages.

## 5.3. Activating Diagnostic Messages

The following environment variables activate diagnostic output:

```
os.environ['ABA_PAMG_DIAGNOSTICS'] = '1'
mp_environment_export=mp_environment_export+('ABA_PAMG_DIAGNOSTICS',)
os.environ['ABA_ EQSITR_LM_INFO '] = 'ON'
mp_environment_export=mp_environment_export+('ABA_EQSITR_LM_INFO',)
```

If you encounter issues that require technical support, collect and send the job files (.amg, .mcp, .log, .mgr, .msg, and .use) together with the input files (if available to share) to your SIMULIA customer support representative.

# 6. References


- SIMULIA User Assistance 2022
  - Abaqus | Analysis | Analysis Procedures | Introduction | Iterative linear equation solver
  - Abaqus | Constraints | Multi-Point Constraints
  - Abaqus | Interactions | Contact Formulations and Numerical Methods | Contact formulations and numerical methods in Abaqus/Standard
  - Abaqus | Analysis | Analysis Solution and Control | Solving nonlinear problems
  - Abaqus | Analysis | Analysis Procedures | Dynamic stress/displacement analysis | Natural frequency extraction
  - Abaqus | Interactions | Contact Difficulties and Diagnostics | Resolving contact difficulties in Abaqus/Standard | Common difficulties associated with contact modeling in Abaqus/Standard
  - Abaqus | Elements | Special-Purpose Elements | Gasket elements | Defining the gasket element's initial geometry

- Knowledge Base Q&A articles
  - QA00000042934: “Using MPI-based parallelization with the Abaqus/Standard direct solver”
  - QA00000053718: “Performance and memory considerations when using the Abaqus/Standard AMG iterative solver”

# 7. Document History

| Document Revision | Date | Revised By | Changes/Notes |
|---|---|---|---|
| 1.0 | 04/14/2022 | Mintae KIM | Original |
| | | | |