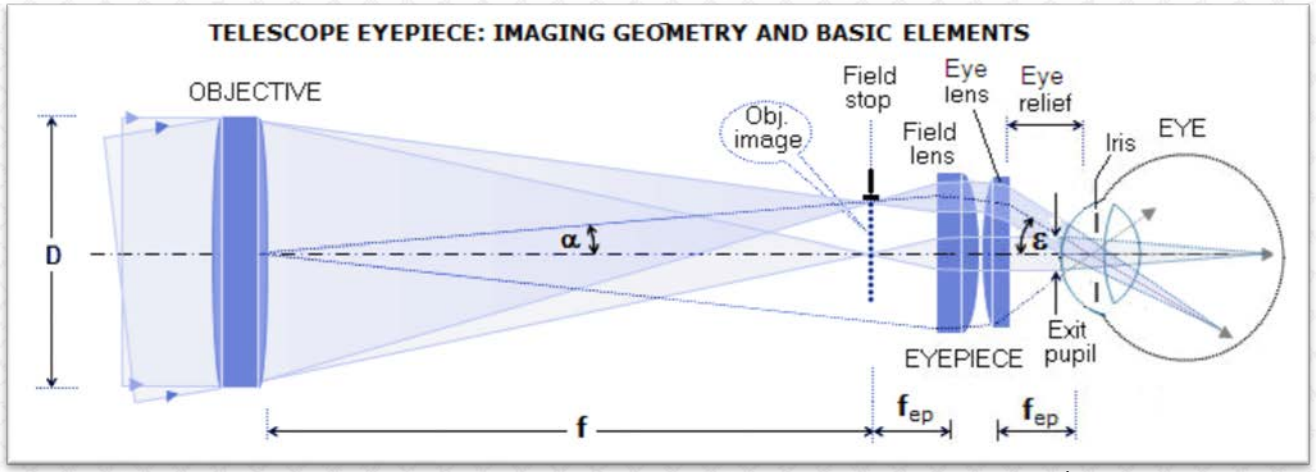
Can ANSYS Mechanical Handle My Required Modeling Precision?

2/8/2017



www.telescope-optics.net

- Recently, PADT received the following inquiry from a scientist interested in using ANSYS for telescope application

Question:
"[Is] it within ANSYS Mechanical's mathematical accuracy on a simple workstation to simulate sub-nanometer deformation of a 1.5m part? (That is better than 1e-10 relative accuracy)"

Let's assume that the relevant ANSYS solvers utilize double precision arithmetic (true for the sparse solver, and is the default setting for pcg). In other words, we may assume that the underlying algebraic equations are solved with double-precision accuracy, and the required results are stored with the same accuracy

One might (naively) assume that, since double precision, 64-bit floating point arithmetic has a relative (rounding) error given by:

$$\varepsilon = \frac{b^{-(p-1)}}{2} = 2 \approx 1.11e-16* \qquad (1)$$

…this implies we can calculate numbers of this size, or equivalently (and perhaps more importantly), that differences between two numbers may be discerned to within quantities as small as this (one definition of precision)

However, this isn't correct!

*where, b is the base (2), p is the number of binary digits in the significand. For 64-bit double precision, this number is 53

The problem is that equation (1) – also called machine precision (or 'machine epsilon') - only gives the *relative error* in an approximation

In other words, if we're trying to approximate a number v, and we have a 64-bit, double precision numerical approximation, v_n, and *if the only source of error is numerical round-off*, then we can expect our approximation to conform to:

$$\varepsilon_R = \left|\frac{v - v_n}{v}\right| \approx 1.11e-16 \qquad (2)$$

But, if we go back to the question, the user appears to define relative accuracy in a curious way:*

$$\varepsilon_A = \frac{v}{L} \approx 1.0e-10 \qquad (3)$$

*"Relative" seems to mean relative to the size, L of the lens instead of the value sought

Unfortunately, the definition (3) is simply incompatible with (1) and (2). At this point, it would be prudent to ask the user to clarify his needs. Nevertheless, for the present purpose, we'll assume that (3) doesn't define a relative error, but rather an *absolute error*:

$$\varepsilon_A = |v - v_n| = 1.0e-10 \quad (4)$$

This assumption allows us to estimate the maximum size of estimates v constrained by our relative numerical error (2):

$$\left|\frac{\varepsilon_A}{v}\right| \geq 1.11e-16$$
$$|v| \leq \sim 1.0e6 \quad (5)$$

So, with the absolute error assumption we made in (4), equation (5) tells us that if we are to remain within the relative error bound set by equation (2), the expected (perhaps analytical) value of estimates, v must be less than around one million. The nanometer-scale estimate (1.0e-9) easily satisfies this requirement.

However, there are a few caveats to keep in mind: We should expect a relative error of the size given in (1) and (2) to occur at each floating point operation (such an error may or may not occur, but we should assume a worst case). Now, the sign of these errors may be positive or negative (numbers get rounded up and down). If they occur randomly, they may cancel one another out. But in the spirit of conservatism, they may also accumulate, in which case, *the actual relative error may be substantially greater than machine precision*. In order to expect a relative error on the order of (1), the total number of floating-point operations must be low

So, if we assume that a positive relative error of 1.11e-16 accumulates with each floating point operation, we reach a cumulative relative error of 1.0e-10 after only one million such operations (this can easily be performed with a medium mesh size). In other words, the cumulative relative error is now equal to the absolute error. We have no more 'wiggle room'.

This possibility places a stricter size-limit on v if the absolute error of 1.0e-10 is to be maintained. Replacing the relative machine error in (5) by the new cumulative error, we have:

$$\left|\frac{\varepsilon_A}{v}\right| \geq 1.0e-10 \qquad (6)$$

$$|v| \leq \sim 1$$

Scaling units may be tried to reduce v in order to conform to the stricter requirement. Notice, however, that there is still no apparent problem with relative error (the upper bound on v is quite reasonable)

But are scaling considerations really necessary? Is it possible to estimate how much cumulative relative error a particular calculation involves? Perhaps not precisely (there will be error in our error estimates!), but the standard estimate in numerical linear algebra (called backward error analysis), yields the following result

$$e_{AE} = e_A c(\mathbf{K}) \qquad (7)$$

Where c(**K**) is the condition number of the assembled system matrix involved in the calculation, ε_{AE} is the absolute expected error, and ε_A is the required absolute error

Now, the condition number of a matrix in ANSYS can be calculated using APDL or APDL Math (for a dense matrix algorithm, see here: http://ansys.net/ansys/tips/acton20080825-condition_number.pdf), or by external code such as Mathematica or Matlab (after exporting the relevant matrices in the appropriate format. See the HBMAT command). However, this can be cumbersome.

An alternative accuracy check may be employed more simply by noting that equation (7) is derived by assuming that the result of a calculation f(x+e) involving perturbed quantity x+e is itself perturbed by the same order of magnitude. In other words:

$$\left| f(x+e) - f(x) \right| \gg |e| \qquad (8)$$

Thus, (8) provides a simple check for whether our model is reliable by replacing e with ε_A

As an example, suppose we're doing a static structural analysis, F = K X. All we have to do is 'perturb' the load by $F+\varepsilon_A$ (note that in this example $f(x+\varepsilon_A) = K^{-1}*(F+\varepsilon_A)$), and check to see if $K^{-1}*(F+\varepsilon_A) - K^{-1}*F \approx \varepsilon_A$.

In fact, we can make this estimate sharper. We can estimate K_{eff} –an effective scalar stiffness associated with the maximum deflection under a prescribed load ($K_{eff} = F/x_{max}$). We choose the load so that the results fall squarely within the display precision of MAPDL or Workbench. With this scalar value, we can then check to see that the following relation holds:

$$x_{\max} = \left\| \mathbf{K}^{-1}(e_A) \right\|_{¥} = \frac{e_A}{K_{eff}} \qquad (9)$$

…and this is the check we've been looking for. Equation (9) says that the maximum displacement result (as indicated by the infinity norm) under a load equal to ε_A, should equal ε_A/k_{eff}

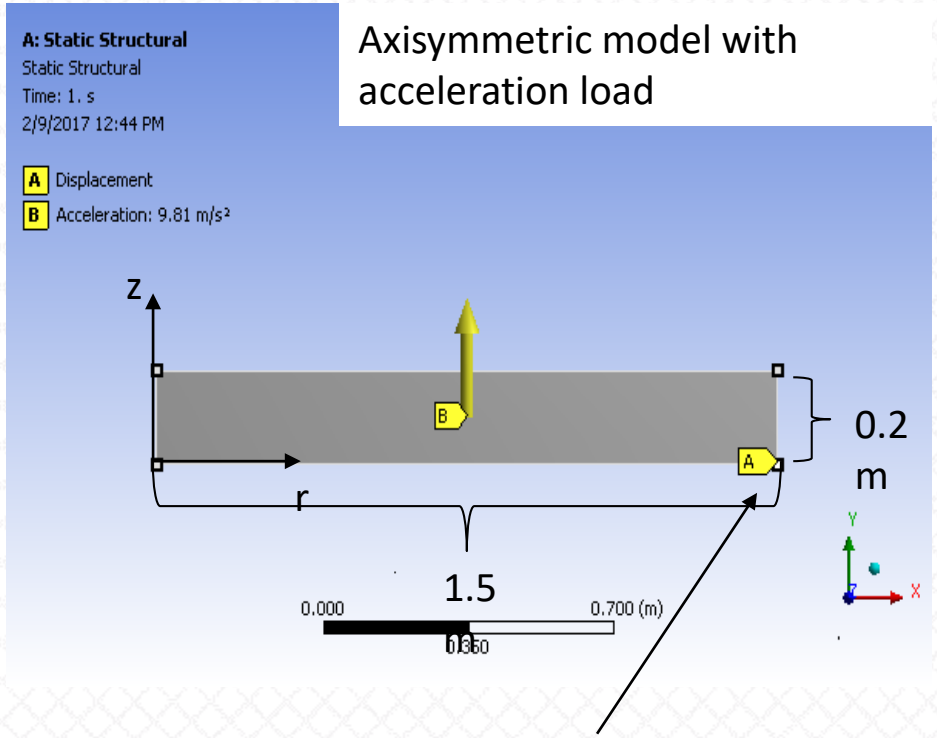
In practice, some care must be taken when implementing equation (9). For a start, because of the problem statement's lack of clarity, we don't know how many significant figures are required in our results. If all ten decimal digits are required, then the only way to do the comparison in (9) is in MAPDL. We would have to get the maximum deflection, store it in a variable, divide the load value by it, and write the result with more than the required significant digits (up to a maximum of 16) to a file.

The reason for this is that variables are only displayed with 9 significant digits in the MAPDL variable viewer. They are displayed with 10 significant digits in the output window. And in WB, results can only be displayed with up to 8 significant digits.

In what follows, we'll assume that six significant digits suffice (in which case we'll have to use scientific notation)

Example

Since the user is analyzing a telescope lens application, we'll create a simplified (flat) lens with a radius of the required 1.5m



Axisymmetric model with acceleration load

This corner fixed in axial direction

ρ = 2399.8 kg/m^3
E = 6.4E10 Pa

Note:
Material properties, loads, and dimensions must all be known with at least the precision required

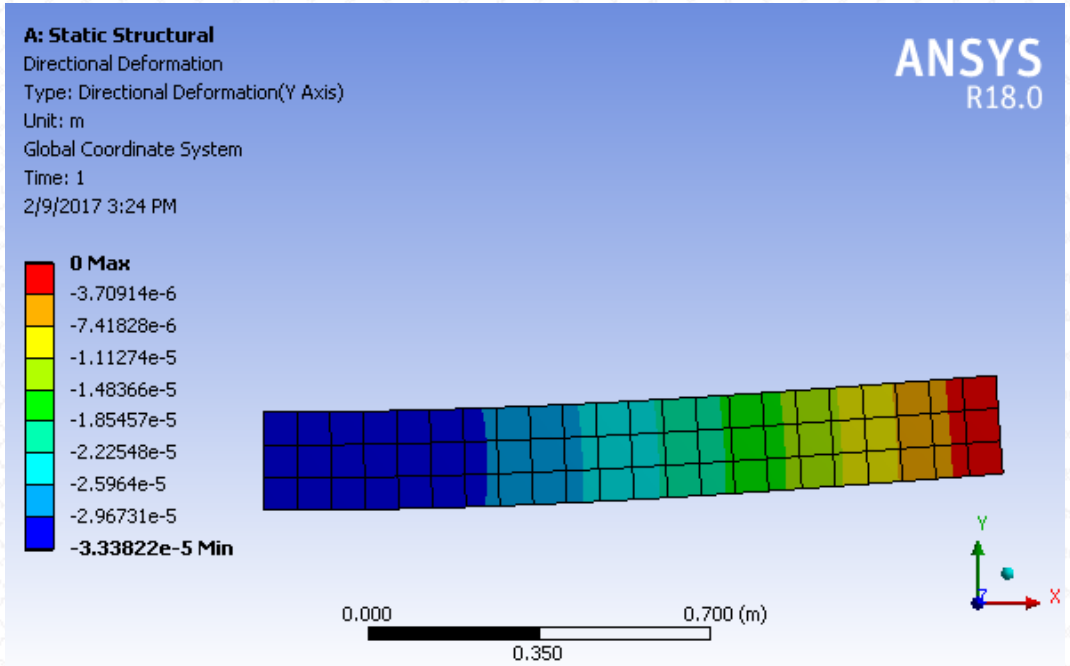
Example

Step 1: Apply a dummy load to extract Keff. In this case, we'll just use 1g. Calculate K_{eff} according to $F/|u_{min}|$

F = ρV= 2.39980e3*9.81*0.2*3.14159*1.5^2
F = 33281.7 kg

$|u_{min}|$ = 3.33822e-5 m

$K_{eff} = F/|u_{min}|$
K_{eff} = 9.96991e9 N/m

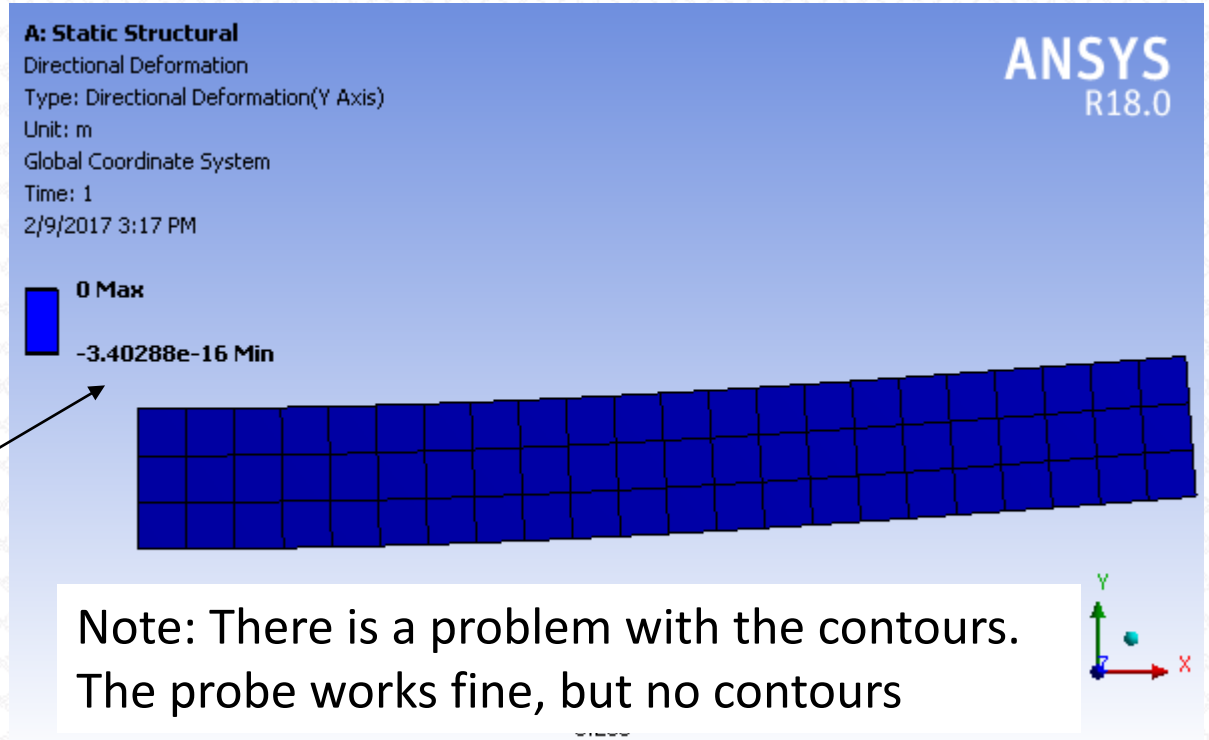


Example

Step 2: Perform a new solve using g = 1.0e-10 m/s^2. Compare the results to ε_A/K_{eff}.

ε_A/K_{eff}=3.40288e-16 m

Results match within significant digits used (assume 6 throughout)

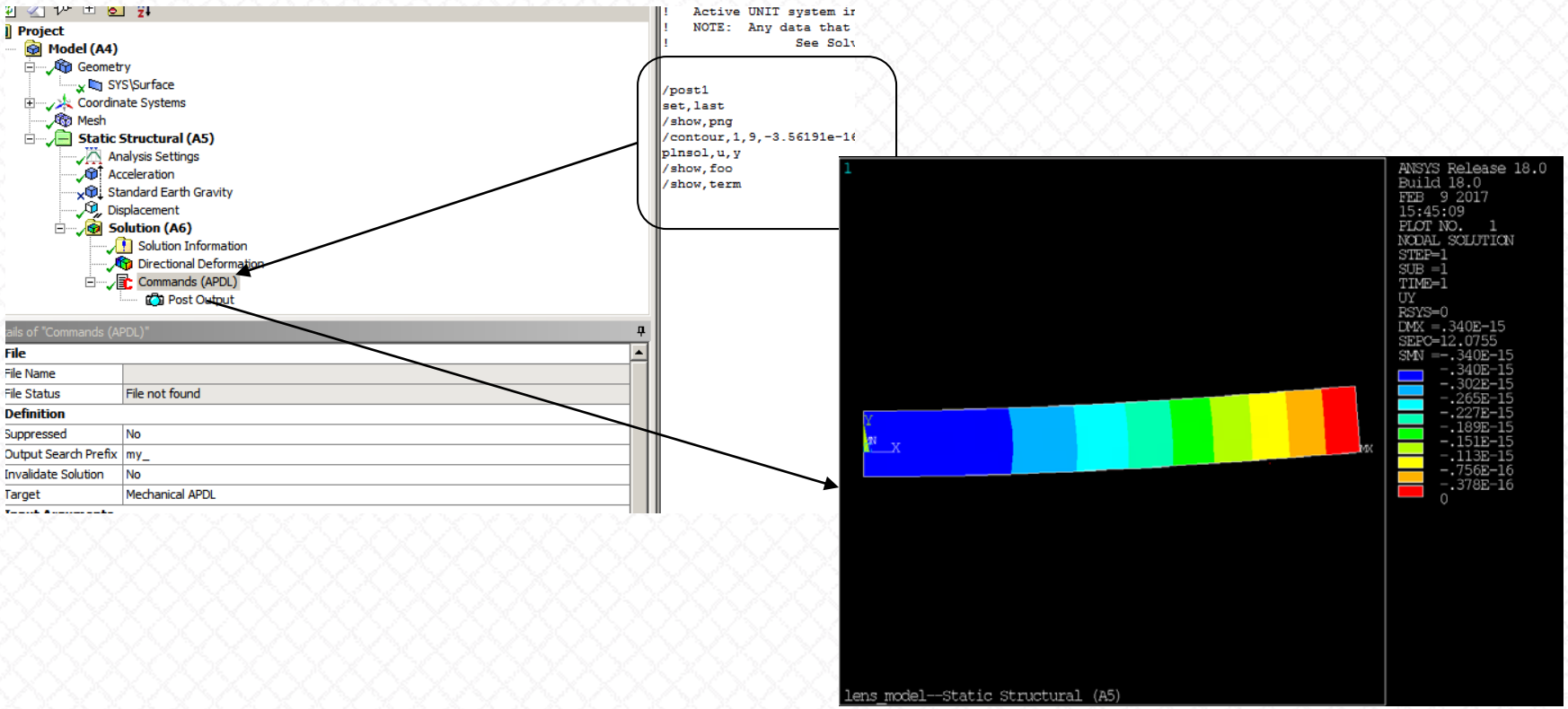


Note: There is a problem with the contours. The probe works fine, but no contours

Example

Step 2 (continued)

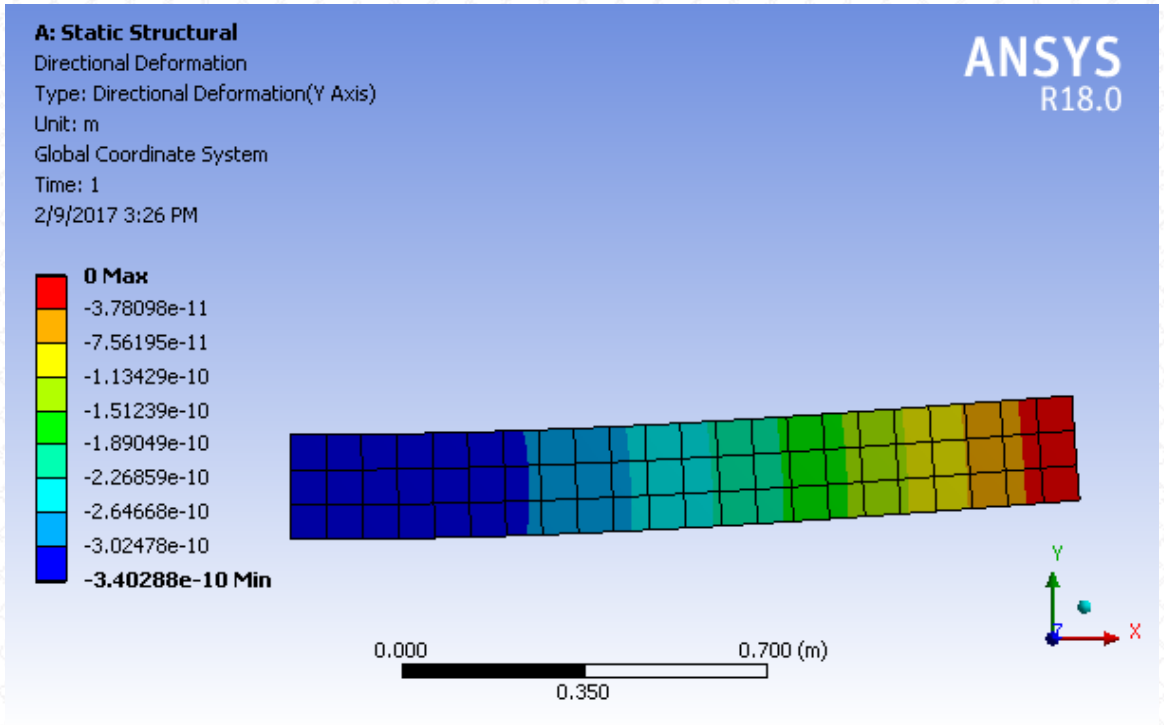
The problem with the WB contour display can be overcome by setting and exporting the required contour range in MAPDL (with a command snippet)



Example

Step 2 (continued)

…OR, we could simply scale the load by some suitable amount. If we scale by 1.0e6, we obtain (note that this trick will not work for nonlinear problems)…



Scaling the load fixes the contour problem in WB.

Conclusions

- Analysts sometimes encounter applications in which, because of the scales involved, questions about the resulting accuracy of simulations arise
- We have provided a simple two-step procedure to determining whether ANSYS can provide reliable solutions in such cases. The steps are:
  1. Apply a dummy load, F to a model in question to extract an effective scalar stiffness value, K_{eff}
  2. Run another simulation applying a load equal to the maximum tolerable absolute error, ε_A. Then compare the min/max result to ε_A/K_{eff}. If they are equal, ANSYS can reliably run the model.
- Depending on the numerical range of results, Workbench may not be able to generate contours (but still produces correct results). If the problem is linear, simply scale the load to an appropriate value to solve this problem.
- If the problem is nonlinear, you will have to post-process in MAPDL
- In the example, a static structural simulation was run. However, the same principles apply to other types of analysis.