Possibilities and drawbacks using arbitrary precision numbers for structural analysis

Simon Klarmann1,* and Jens Wackerfuß2
1 Chair of Structural Analysis and Dynamics, RWTH Aachen University, Germany
2 Institute of Structural Analysis, University of Kassel, Germany

In various areas of computational mechanics, rounding errors can have a considerable influence on the quality of the simulation results; in some cases, these lead to the termination of the numerical calculation. Rounding errors are caused by limited accuracy in the representation of floating point numbers. Current codes usually use double precision numbers (p=16 significant digits). Until now, modern multi-precision libraries with which floating-point numbers can be processed with arbitrary accuracy are largely unused.

The aim of this article is to show the possibilities and limitations of such libraries in the context of computational mechanics. The accuracy of computations from p=8 up to p=128 will be investigated. Examples will be selected which are particularly sensitive to rounding errors. On the basis of a first academic example it is examined which calculation accuracy is necessary to carry out a static analysis on a cantilever beam with a slenderness of up to 1049 with a standard beam FE formulation. In a second example, a load-bearing structure is analyzed in which the stiffness of its supporting members differs by several powers. Finally, the disadvantages associated with the higher calculation accuracy (CPU time, memory requirements) are discussed.

1 Introduction

Increasing computational power not only enables us to compute larger models, but allows to do the analysis with higher number accuracy. Modern tools like the Eigen C++ library [1] offer easy access to perform dense and sparse linear algebra. The library includes sparse matrix solvers, too. Furthermore, its templated implementation offers the possibility to use any kind of number type. In combination with Boost’s [2] multiprecision number types it is possible to perform calculations with arbitrary precision. To do so, the cpp_bin_float type is chosen. It represents the numbers with fixed-precision, but the amount of non-zero digits can be set arbitrarily at compile time.

2 Theory

When performing finite element analysis, in most of the cases the necessity to solve sparse linear systems of equations arises:

\[ \mathbf{K} \mathbf{u} = \mathbf{f}. \] (1)

The system is given in eq. (1), with the stiffness matrix \( \mathbf{K} \), the solution vector \( \mathbf{u} \) and the load vector \( \mathbf{f} \). As in computer simulations the number of digits to represent their values is limited, the solution of eq. (1) is affected by round-off errors. According to [3], a conservative estimate on the number of correct digits in the solution is:

\[ s \geq p - \log[c(\mathbf{K})]. \] (2)

Herein, \( s \) is the number of correct digits in the solution, \( p \) the number of digits each value in eq. (1) is represented, \( c(\mathbf{K}) \) is the condition number of the matrix \( \mathbf{K} \) and \( \log \) the logarithm to the base 10. The condition number itself is defined as the maximum eigenvalue \( \lambda_{\text{max}} \) divided by the minimum eigenvalue \( \lambda_{\text{min}} \) of \( \mathbf{K} \). As shown in [3], a better estimate of the numerical accuracy can be obtained, if the condition number \( c(\mathbf{K}) \) of the scaled matrix \( \hat{\mathbf{K}} \) is used in eq. (2). A common way of scaling the matrix with its diagonal elements is proposed

\[ \hat{\mathbf{K}} = \mathbf{D} \mathbf{K} \mathbf{D}, \quad \text{with} \quad d_{ii} = \frac{1}{\sqrt{k_{ii}}}. \] (3)

Herein, \( d_{ii} \) are the diagonal entries of \( \mathbf{D} \) and \( k_{ii} \) the ones of \( \mathbf{K} \).

3 Numerical Example

As said, \( p \) represents the number of digits to store the floating-point values, e.g. \( p = 8 \) is the basic single precision and \( p = 16 \) the basic double precision number type. For higher order precision \( p \geq 32 \) Boost’s cpp_bin_float is used.
### 3.1 Numerical Error

To investigate the impact on the numerical error, a simple LSFEM formulation for Bernoulli’s beam theory is chosen. Firstly, the differential equation is transformed into a system of first order differential equations and then the squared L2-norm of each equation gets summed up. Finally, this sum gets minimized:

\[
G(w, \beta, M, Q) = \|w' - \beta\|^2_0 + \|EI\cdot\beta' - M\|^2_0 + \|M' - Q\|^2_0 + \|Q'\|^2_0 \rightarrow \text{min}.
\]  

(4)

To set up the equation system, the variation of eq. (4) is determined. Afterwards each function (vertical displacements \(w\), rotations \(\beta\), moments \(M\) and shear force \(Q\)) is discretized using linear Lagrange shape functions. The modelled system is depicted in fig. 1. Figure 2 shows the dependence between the condition number and the total number of degrees of freedom (Total Dof). As one can observe, the condition number of the scaled matrix \(c(\hat{K})\) is several orders of magnitude lower than the one of the original matrix \(c(K)\). When comparing the solution error with \(p = 16\) in fig. 3, with the results in fig. 2, it is clearly visible that using the scaled matrix to compute the condition number a better accuracy estimate is achieved. At the same time, this condition number reaches a value of approx. \(10^{16}\), the error curve starts to deteriorate. In this case, increasing the number of digits, to e.g. \(p = 32\) recovers the convergent behavior of the solution.

### 3.2 Performance

In this example the computational cost is investigated. Using direct solvers, the factorization of the matrix is in general the most time consuming part of the calculation. Therefore, the time in case of LL\(^T\)-factorization of Eigen’s SimplicialLLT solver is observed. The system used is depicted in fig. 4. It is a simple cantilever beam modelled with triangular shell elements using standard finite elements. In fig. 5 the factorization time over the number of degrees of freedom for different number precision is shown. In case of approximately \(3\cdot10^5\) unknowns, switching from the native number type to the cppBinFloat number type seems to have a huge impact on the performance as the time for the LL\(^T\) factorization increases by a factor of approximately 30, see fig. 6. In contrast to that, switching from single precision to double precision approximately doubles the time needed for factorization. The same happens when switching from \(p = 32\) to \(p = 64\) or \(p = 64\) to \(p = 128\).

**Acknowledgements**  Open access funding enabled and organized by Projekt DEAL.

**References**

[1] Eigen C++ library, http://eigen.tuxfamily.org/, last visited: 19.5.2020

[2] Boost C++ libraries, https://www.boost.org/, last visited: 19.5.2020

[3] H. Lindenberg Neto, Advances in Engineering Software 19:3, 129–142 (1994).