Generation of Sparse Jacobians for the Function Mock-Up Interface 2.0

J. Åkesson[a,c], W. Braun[d], P. Lindholm[b], B. Bachmann[d]
[a]Lund University, Department of Automatic Control, Lund, Sweden
[b]Lund University, Department of Mathematics, Lund, Sweden
[c]Modelon AB, Lund, Sweden
[d]University of Applied Sciences Bielefeld, Bielefeld, Germany

Derivatives, or Jacobians, are commonly required by numerical algorithms. Access to accurate Jacobians often improves the performance and robustness of algorithms, and in addition, efficient implementation of Jacobian computations can reduce the over-all execution time. In this paper, we present methods for computing Jacobians in the context of the Functional Mock-up Interface (FMI), and Modelica. The algorithmic machinery employed consists of known methods and algorithms, such as numerical, symbolic, and automatic differentiation, as well as graph theoretic methods such as the BLT transformation. Two prototype implementations, sharing similarities as well as differences have been presented. One of the methods is a straight forward application of forward automatic differentiation and generation of C code, which results in functions for evaluation of directional derivatives, which in turn are used to compute Jacobians. The other method relies mainly on symbolic differentiation and makes use of symbolic simplification algorithms in a Modelica compiler to generate directional derivative functions. Both methods provide sparsity patterns for the ODE Jacobians, and and they both make efficient use of sparsity in order to reduce the number of directional derivative evaluations, a techinque referred to as compression.

The two approaches are implemented in JModelica.org and OpenModelica, respectivly, and compared in an industrial benchmark as well as in several synthetic benchmarks. Both implementations show linear growth in key measures such as model compilation time, generated code size and execution time, under realistic assumptions on model structure. In terms of execution speed, the method relying on symbolic differentiation and symbolic processing, as implemented in OpenModelica, performed faster.

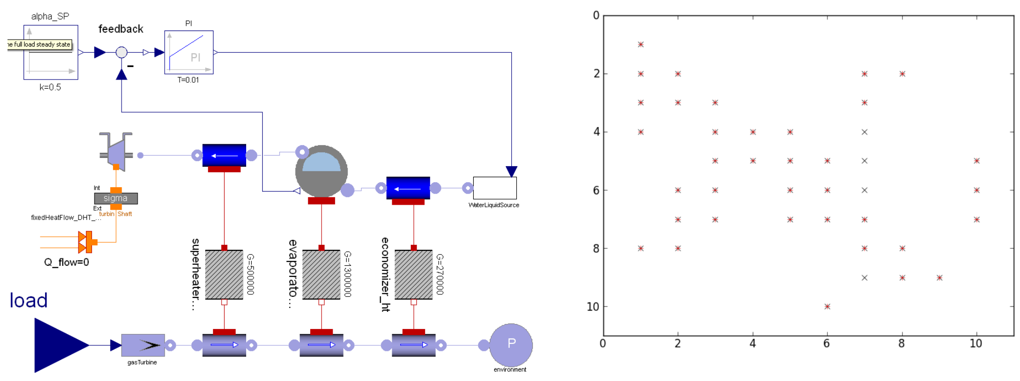

Figure 1: A power plant model used as benchmark in the papers, and the corresponding sparsity pattern.