Numerical Libraries

Advantage: Development of primal code
- Save development time
- Concentrate on the research
- Write robust code

Problem: Development of adjoint code
If the library does not provide adjoint versions of the routines user has to
- Write continuous adjoint version of the routines as described in [3] or [4]
  - requires deeper insight into algorithm
  - cost a lot of development time
- Use AD tool
  - source of the routine must be available
  - big code base
- Structure of the library code does not fit with requirements of the AD tool

Solution
The adjoint routines should be provided by the vendor of the library as has
- the best knowledge and expertise about the algorithms
- access to the source code
- testing infrastructure (time saving)
- providing different adjoint versions
  - Algorithmic Adjoint (AA): Derivative of primal algorithm with machine accuracy through plain application of local adjoint code generation rules.
  - Robust Algorithmic Adjoint (RoAA): Robust version of AA implementing, for example checkpointing and/or preaccumulation techniques. (Different versions are possible.)
  - Symbolic Adjoint (SA): Derivative under certain assumptions about availability of exact primal solution, for example based on the implicit function theorem, see Griewank and Walther (2008). (Different versions are possible.)
  - Hybrid Algorithmic / Symbolic Adjoint (HASA): Combination of AA/RoAA and SA. (Different versions are possible.)

NAG AD Library

The NAG AD Library contains adjoint version of the NAG Library routines using C++ AD overload

The AD version of the routines is generated by replacing all floating point variables in the code with the first order adjoint variables from dco/c++ or dco/fortran (see Listing 2). We refer to the resulting interface as association by address interface (the reason for the name is that the value and the adjoint part are distinguished through the address). dco/fortran uses the same dco/c++ tape backend, allowing the user to mix C++ and Fortran code calling NAG Routines, writing in the same tape backend just by replacing datatypes.

For those who do not use these AD tools we provide the association by name interface (see Listing 3). The association by name interface internally is based on a driver that calls the association by address interface, extracts the desired adjoint projections and stores them in the corresponding adjoint variables. The adjoint variables are identified by name suffix _adj. This is also the reason for the naming, the value and adjoint information is distinguished by name.

References