Elements of Design for Containers and Solutions in the LinBox library

Brice Boyer[1], Jean-Guillaume Dumas[2], Pascal Giorgi[3], Clément Pernet[4], and B. David Saunders[5]

[1] North Carolina State University, Department of Mathematics, Raleigh, NC, USA[†], bbboyer@ncsu.edu.
[2] Laboratoire J. Kuntzmann, Université de Grenoble. 51, rue des Mathématiques, umr CNRS 5224, bp 53X, F38041 Grenoble, France[‡], Jean-Guillaume.Dumas@imag.fr.
[3] Laboratoire d'Informatique, de Robotique et de Microélectronique de Montpellier (LIRMM), CNRS, Université Montpellier 2, 161 rue ADA, F-34095 Montpellier, France[‡], pascal.giorgi@lirmm.fr.
[4] Laboratoire LIG, Université de Grenoble et INRIA. umr CNRS, F38330 Montbonnot, France[‡], clement.pernet@imag.fr.
[5] University of Delaware, Computer and Information Science Department. Newark / DE / 19716, USA, saunders@udel.edu.

Abstract. We develop in this paper design techniques used in the C++ exact linear algebra library LinBox. They are intended to make the library safer and easier to use, while keeping it generic and efficient.
First, we review the new simplified structure of the containers, based on our *founding scope allocation* model (cf. [1]). Namely, vectors and matrix containers are all templated by a field and a storage type. Matrix interfaces all agree with the same minimal blackbox interface. This allows e.g. for a unification of our dense and sparse matrices, as well as a clearer model for matrices and submatrices. We explain the design choices and their impact on coding. We will describe serveral of the new containers, especially our sparse and dense matrices storages as well as their `apply` (*blackbox*) method and compare to previous implementations.
Then we present a variation of the *strategy* design pattern that is comprised of a controller–plugin system: the controller (solution) chooses among plug-ins (algorithms) and the plug-ins always call back the solution so a new choice can be made by the controller. We give examples using the solution `mul`, and generalise this design pattern to the library. We also show performance comparisons with former LinBox versions.
Finally we present a benchmark architecture that serves two purposes. The first one consists in providing the user with an easy way to produces graphs using C++. The second goal is to create a framework for automatically tuning the library (determine thresholds, choose algorithms) and provide a regression testing scheme.

[†] This material is based on work supported in part by the National Science Foundation under Grant CCF-1115772 (Kaltofen)
[‡] This material is based on work supported in part by the Agence Nationale pour la Recherche under Grant ANR-11-BS02-013 HPAC (Dumas, Giorgi, Pernet).

References

1. J.-G. Dumas, T. Gautier, C. Pernet, and B. D. Saunders. LinBox founding scope allocation, parallel building blocks, and separate compilation. In K. Fukuda, J. Van Der Hoeven, and M. Joswig, editors, *Proceedings of the Third International Congress Conference on Mathematical Software*, volume 6327 of *ICMS'10*, pages 77–83, Berlin, Heidelberg, Sept. 2010. Springer-Verlag.