1

Object-Oriented Numerics

Erlend Arge¹
Are Magnus Bruaset²
Hans Petter Langtangen³

ABSTRACT This chapter is concerned with the use of object-oriented programming techniques for numerical applications, especially in terms of the computer language C++. Through a series of examples we expose some of the strengths and possibilities of object-oriented numerics.

1.1 Introduction

Many fields of science rely on various types of mathematical models, typically used to describe dynamic processes in nature or for representation and analysis of information gathered from measured data sets. In most applications dealing with such models, computers are necessary tools in order to convert the researcher’s intuition and experiences, via critical hypotheses and complicated equations, into numbers indicating success or failure. Throughout history, ranging from the innovative mechanical devices designed by Babbage and Newton to the turbo-charged electronical wonders of today, the need for doing numerical computations has strongly influenced the development of computing machinery. Today, we are routinely solving problems that would be impossible to attack with the help of pen, paper and brain alone. As the performance of computers increases, we will continue to stretch the limits of what is accepted to be computationally feasible enterprises. This thrilling development also puts extreme demands on quality management and robustness at all levels of the problem solving process. In particular, we need flexible but rigorous techniques and guidelines for software development.

Traditionally, numerical software development is based on use of proce-
dural languages like Fortran and C. In large applications, the involved procedures are wrapped up in libraries, possibly also linking to other external libraries like the well-known BLAS routines for basic linear algebra operations [17]. These principles for software development have remained more or less static over the last decades, partly because of inherent limitations of simple procedural languages. Over the last few years, there has been an increasing interest in applying the paradigms of object-oriented programming (OOP) in numerical software development [2,9,10,22]. This approach seems attractive due to well-defined mechanisms for modular design, re-use of code and for creating flexible applications that can be easily extended with respect to problem classes as well as solution methods. In short, OOP encourages computer implementations of mathematical abstractions. This statement is indeed supported by practical experiences made in the Diffpack [8] and Siscat [3] projects, respectively utilizing OOP for software development in the fields of partial differential equations and approximation of scattered data. Throughout this presentation we will use the term object-oriented numerics (OON) to denote the use of object-oriented programming techniques in numerical applications.

Hopefully, this chapter will reveal that OOP requires a new way of thinking about programming. At first sight, object-oriented implementations may often be more abstract and more difficult to understand than conventional codes. Usually, Fortran programmers need several months of practice before they will manage to improve the quality of their applications using OON. It is therefore natural to ask the question: Who needs object-oriented numerics? Modern scientific computing imposes two important demands on the implementation; those regarding efficiency and complexity. Efficiency typically consists of CPU-time and human time, the latter reflecting the work associated with implementation, testing and maintenance. The goal of OON is to keep the CPU-time constant while reducing the human time drastically. In addition, OON addresses complicated problems. The complexity can be on both the algorithmic and the application side. The building blocks of OON support a step-wise software development from simple problems to more complicated problems in a unified framework. The key to this goal is the reuse of already tested software components, which is a fundamental issue in OOP. Interfacing to other software packages is also made simpler by an object-oriented design. Finally, we will mention a very important requirement of software used for research, namely flexibility in the choice of problems and methods. The real advantage of OOP will in many cases be the possibility of building flexible codes for scientific experimentation. To summarize, comprehensive and complicated physical problems or numerical algorithms are candidates for object-oriented implementations. If your problems are solved by a few thousand lines of clean Fortran or C code, OON will probably not yield a substantial gain in human efficiency.

Although Fortran and C can be used to implement object-oriented designs [16,19], the period of development is significantly reduced when ap-