Chapter 3

Numerical Linear Algebra

This chapter aims to provide a brief introduction to numerical linear algebra. People who are unfamiliar with how to go about (say) solving linear equations, or how to compute eigenvalues and eigenvectors might find this useful for selecting the best routine(s) to solve their particular problem, and to understand the rationale for the way the routines are set up in the way they are.

3.1 What numerical linear algebra is about

There are a number of core operations and tasks that make up numerical linear algebra. At the lowest level these include calculating linear combinations of vectors and inner products, and at the higher level consists of solving linear equations, solving least-squares problems and finding eigenvalues and eigenvectors.

The lower level operations are usually quite straightforward in terms of what they do and what the accuracy of the results are. However, with higher level operations more care must be taken with regard to both efficiency and the accuracy of the answers. The routines used to perform these higher level operations are more varied and allow a number of different ways of performing the same computation. The difference between them lies often in the speed (or lack of it) and the accuracy of the answers obtained.

There are further complications because of some intrinsic limits to the computations that a computer can do accurately, at least with floating point arithmetic. Floating point arithmetic cannot store numbers to an accuracy (relative to the number stored) better than what is called "*machine epsilon*", or "*unit roundoff*". This quantity is usually denoted by `u`, but is represented in the library by **`MACHEPS`**. It is also referred to in the ANSI C header file `<float.h>` as **`DBL_EPSILON`** for double precision and **`FLT_EPSILON`** for single precision. For most machines this quantity is about 2×10^{-16} for double precision, and 10^{-7} for single precision.

Practically all floating point calculations introduce errors of size of machine epsilon times the size of the quantities involved; for all intents and purposes, these errors are unavoidable. Perturbations in the *data* of a problem are essentially unavoidable. Algorithms that compute answers that would be exact for slightly perturbed data