Meschach

Matrix Computations in C

Introduction

Most of numerical analysis relies on algorithms for performing caclulations on matrices and vectors. The operations most needed are ones which solve systems of linear equations, solve least squares problems, and eigenvalue and eigenvector calculations. These operations form the basis of most algorithms for solving systems of nonlinear equations, numerically computing the maximum or minimum of a function, or solving differential equations.

The Meschach library contains routines to address all of the basic operations for dealing with matrices and vectors, and a number of other issues as well. I do not claim that it contains every useful algorithm in numerical linear algebra, but it does provide a basis on which to build more advanced algorithms. The library is intended for people who know something of the 'C' programming language, something of how to solve the numerical problem they are faced with (which involves matrices and/or vectors) but don't want to have the hassle of building all the necessary operations from the ground up. I hope that researchers, mathematicians, engineers and programmers will find this library makes the task of developing and producing code for their numerical problems easier, and easier to maintain than would otherwise be possible.

To this end the source code is available to be perused, used and passed on without cost, while ensuring that the quality of the software is not compromised. The software *is* copyrighted; however, the copyright agreement follows in the footsteps of the Free Software Foundation in preventing abuse that occurs with totally "public domain" software.

This is not the first or only library of numerical routines in C. However, there are still a number of niches which have not been filled. Some of the currently available libraries are essentially translations of Fortran routines into C. Those that attempt to make use of C's features usually address a relatively small class of problems. There is a commercial package of C++ routines (and classes) for performing matrix computations, and NAG and IMSL are producing C versions of their libraries, but none of these is "public domain" in any sense. The Meschach library makes extensive use of C's special features (pointers, memory allocation/deallocation, structures/records, low level operations) to ease use and ensure good performance. In addition, Meschach addresses the need for both dense and sparse matrix operations within a single framework.

There is another issue which needs to be addressed by a matrix library like this. At one end, libraries that are essentially translations from Fortran will make little use of memory allocation. At the other end, interactive matrix "calculators" such as MATLAB and MATCALC use memory allocation and garbage collection as a matter of course and have to interpret your "program". This latter approach is very flexible, but resource hungry. These matrix calculator programs were not designed to deal with large problems.

This matrix library is intended to provide a "middle ground" between efficient but inflexible Fortran-style programs, and flexible but resource hungry calculator/interpreter programs. When and how memory is allocated in Meschach can be controlled by using the allocation/deallocation and resizing routines; result matrices and vectors can be created dynamically when needed, or allocated once, and then used as a static array. Unnecessary memory allocation is avoided where necessary. This means that prototyping can often be done on MATLAB or MATCALC, and final code can be written that is efficient and can be incorporated into other C programs and routines without having to re-write all the basic routines from scratch.

Finally, I would like to thank all those at the University of Queensland Mathematics Department, at Opcom, and at the Australian National University for their interest in and comments on this matrix library. In particular, I would like to thank Martin Sharry, Michael Forbes, Phil Kilby, John Holt, Phil Pollett and Tony Watts at the University of Queensland, and Mike Osborne at the Australian National University and Karen George from the University of Canberra.

David E. Stewart, Canberra, Australia, 1992

Meschach

Matrix Computations in C

1 Tutorial **1**
- 1.1 The data structures and some basic operations . . . 1
- 1.2 How to manage memory . . . 4
  - 1.2.1 No deallocation . . . 4
  - 1.2.2 Allocate and deallocate . . . 4
  - 1.2.3 Resize on demand . . . 5
- 1.3 A routine for a 4th order Runge–Kutta method . . . 5
- 1.4 A least squares problem . . . 10
- 1.5 A sparse matrix example . . . 12
- 1.6 How do I ....? . . . 14
  - 1.6.1 .... solve a system of linear equations . . . 14
  - 1.6.2 .... solve a least-squares problem . . . 14
  - 1.6.3 .... find all the eigenvalues (and eigenvectors) of a general matrix . . . 14
  - 1.6.4 .... solve a large, sparse, positive definite system of equations . . . 15

2 Data structures **16**
- 2.1 Vectors . . . 16
  - 2.1.1 Integer vectors . . . 17
- 2.2 Matrices . . . 17
- 2.3 Permutations . . . 18
- 2.4 Basic sparse operations and structures . . . 18
- 2.5 The sparse data structures . . . 19
- 2.6 Sparse matrix factorisation . . . 21
- 2.7 Iterative techniques . . . 22
- 2.8 Other data structures . . . 23

3 Numerical Linear Algebra **24**
- 3.1 What numerical linear algebra is about . . . 24
- 3.2 Vector and matrix norms . . . 25
- 3.3 "Ill conditioning" or intrinsically bad problems . . . 26
- 3.4 Least squares and pseudo-inverses . . . 26
  - 3.4.1 Singular Value Decompositions . . . 27
  - 3.4.2 Pseudo-inverses . . . 27
  - 3.4.3 QR factorisations and least squares . . . 28
- 3.5 Eigenvalues and eigenvectors . . . 29
- 3.6 Sparse matrix operations . . . 31

4 Basic Dense Matrix Operations **32**

5 Dense Matrix Factorisation Operations **70**

6 Sparse Matrix Operations **93**

7 Installation and copyright **120**
7.1 Installation . . . 120
7.1.1 makefile . . . 121
7.1.2 machine.h . . . 121
7.1.3 machine.c . . . 122
7.2 Copyright . . . 123

8 Designing numerical libraries in C **125**
8.1 Numerical programming in C . . . 125
8.1.1 On efficient compilers . . . 125
8.1.2 Strategies for using C . . . 126
8.1.3 Non-C programmers start here! . . . 127
8.2 The data structures . . . 129
8.2.1 Pointers to struct's . . . 130
8.2.2 Really basic operations . . . 130
8.2.3 Output . . . 132
8.2.4 Copying . . . 133
8.2.5 Input . . . 133
8.2.6 Resizing . . . 135
8.3 How to implement routines . . . 135
8.3.1 Design for debugging . . . 135
8.3.2 Workspace . . . 136
8.3.3 Where to put the output . . . 138
8.4 User-defined functions . . . 139
8.5 Building the library . . . 141
8.5.1 Numerical aspects . . . 141
8.6 Debugging . . . 142
8.6.1 Memory allocation bugs . . . 142
8.6.2 If all else fails . . . 143
8.7 Suggestions for enthusiasts . . . 143
8.8 Pride and Prejudice . . . 143
8.8.1 Why don't I use `float` instead of `double`? . . . 143
8.8.2 What about Fortran 90? . . . 144
8.8.3 Why should people writing numerical code care about good software? . . . 144