clem: A common-lisp matrix package

Cyrus L. Harmon

April 23, 2007

1 Abstract

CLEM is an open-source Common Lisp library for the representation and manipulation of matrices. CLEM is designed to be a flexible and extensible system for the representation of arbitrary 2-dimensional matrices.

2 Introduction

The Common Lisp language[1] offers a rich, dynamic environment for programming and data analysis. Common Lisp contains a powerful object system, the Common Lisp Object System (CLOS)[2], and most modern implementations support a protocol for the generation not just of new classes and objects, but to extend the object system itself using the Meta-object Protocol[3].

CLEM uses CLOS and the Meta-object protocol (MOP) to define astandard-matrix-class that serves as the metaclass for classes that represent matrices with elements of specific types. The typed matrices can represent matrices containing values of specific types in the Common Lisp type system, starting with type t as the most general data type, and becoming more restrictive by using more specific types suchdouble-float, fixnum, or (unsigned-byte 8). By using the most specific type that can represent the values of a given matrix, the lisp system can optimize for better performance and memory usage requirements. For example, a bit-matrix will use 1 bit per matrix element, rather than 32-bits on 32-bit systems for a t-matrix.

2.1 Matrix Types

2.2 Matrix Representation

Common Lisp provides a rich built-in array type which serves as the storage for CLEM matrices. Given that Common Lisp has built-in arrays, why do we need CLEM and what value is provided by creating a set of classses around arrays? First, the Common Lisp arrays have a limited set of operations defined on them. While there is a built-in (scalar) addition operator, there is no built-in way to perform an element-wise addition of two arrays. CLEM addresses these by defining a set of generic functions that operate on matrices that provide a number of commonly used matrix operations such as matrix arithmetic. Second, there is no way to define methods on arrays based on their element types. Therefore, we define subclasses of matrix whose underlying arrays are specialized to distinct types. We can then define methods to operate specifically on these subclasses, affording the opportunity to treat, say, floating point and integer matrices differently and to provide declarations to the compiler based on the array element type, which can, in Common Lisp implementations with sufficiently smart compilers, lead to much improved performance.

3 Defining CLEM Classes and Making CLEM Instances

3.1 Creating CLEM Instances with make-instance

The following code creates a 16-row by 16-column matrix of typedouble-float-matrix and assigns it to the dynamic variable*m1*.

```
(defparameter *m1* (make-instance 'double-float-matrix :rows 16 :cols 16))
*M1*

*m1*
#<DOUBLE-FLOAT-MATRIX [.000000000 .000000000 .000000000 .000000000 .000000000 .000000000 .000000000 ...
 .000000000 .000000000 .000000000 .000000000 .000000000 .000000000 .000000000 ... .000000000;
 .000000000 .000000000 .000000000 .000000000 .000000000 .000000000 .000000000 ... .000000000;
 .000000000 .000000000 .000000000 .000000000 .000000000 .000000000 .000000000 ... .000000000;
 .000000000 .000000000 .000000000 .000000000 .000000000 .000000000 .000000000 ... .000000000;
 .000000000 .000000000 .000000000 .000000000 .000000000 .000000000 .000000000 ... .000000000;
 .000000000 .000000000 .000000000 .000000000 .000000000 .000000000 .000000000 ... .000000000;
 ...
 .000000000 .000000000 .000000000 .000000000 .000000000 .000000000 .000000000 ... .000000000]>
```

The default is to only show the first 7 and the last rows and columns of each matrix. The number of rows and columns can be changed by setting the *matrix-print-row-limit* and*matrix-print-col-limit* variables.

3.2 standard-matrix-class

3.3 CLEM Matrix Types

3.3.1 Number matrices

The most general class of numerical matrix is the number matrix.

3.3.2 Integer Matrices

3.3.3 Floating-point Matrices

3.3.4 Complex-value Matrices

4 Working with CLEM Matrices

4.1 Matrix Dimensions and Values

4.2 Typed matrix operations

4.3 Matrix Copying

4.4 matrix-move

5 Matrix Arithmetic

5.1 Matrix Addition and Subtraction

5.2 Matrix Multiplication

5.3 Hadamard Product

5.4 Scalar Arithmetic

5.5 Other Mathematical Functions

Discuss mat-log, mat-abs, min, and max.

6 Matrix Operations

6.1 Matrix Inversion

6.2 Matrix Normalization

6.3 Discrete Convolution

6.3.1 Derivatives

6.3.2 Gradient Magnitude

6.3.3 Gaussian Blur

6.4 Affine Transformations

6.4.1 Interpolation

6.5 Morphological Operations

6.5.1 Dilation and Erosion

6.5.2 Variance

6.5.3 Thresholding

7 CLEM Implementation Details

7.1 Type-specific matrix functions

The general strategy has been to 1) make things work and then make them work quickly. To this end, I have been writing functions for matrix operations in a general manner first and then recoding type-specific versions to make certain operations go faster. This is done via liberal use of macros to generate type-specific functions and methods for matrix operations that go much faster than the general versions.

The convention is that a generic function such as sum-range will have a generic version that works with all matrices and type specific versions thaqt work with specific matrices. g In order to support these functions there may be internal methods, prefixed with a functionality. Macros that generate the code used for the type-specific methods will be prefixed with a the generate in-place code where the overhead of the method-call to the widely enforced and certainly untested. Hopefully this situation will improve.

7.2 Hacking the SBCL compiler to improve performance

References

[1] G. L. Steele, Jr., *Common Lisp: the Language* (Digital Press, Bedford MA, 1990), second edn.

[2] S. E. Keene, *Object-Oriented Programming in Common Lisp: A Programmer's Guide to CLOS* (Addison-Wesley Professional, 1989).

[3] G. Kiczales, *The Art of the Metaobject Protocol* (The MIT Press, 1991).