UMFPACK and BOOST's uBLAS Sparse Matrix

Asked 14 years, 2 months ago Modified 14 years, 1 month ago

Viewed 4k times

4

I am using Boost's uBLAS in a numerical code and have a 'heavy' solver in place:

http://www.crystalclearsoftware.com/cgi-bin/boost_wiki/wiki.pl?LU_Matrix_Inversion

The code works excellently, however, it is painfully slow. After some research, I found UMFPACK, which is a sparse matrix solver (among other things). My code generates large sparse matrices which I need to invert very frequently (more correctly solve, the value of the inverse matrix is irrelevant), so UMFPACk and BOOST's Sparse_Matrix class seems to be a happy marriage.

UMFPACK asks for the sparse matrix specified by three vectors: an entry count, row indexes, and the entries. (See example).

My question boils down to, can I get these three vectors efficiently from BOOST's Sparse Matrix class?

c++ boost linear-algebra numerical umfpack

Share

Improve this question

Follow

edited Oct 22, 2010 at 12:10

asked Oct 21, 2010 at 15:26

ccook

5,939 ● 6 ● 57 ● 82

1 Answer

Sorted by: Highest score (default)

6

There is a binding for this:

http://mathema.tician.de/software/boost-numeric-bindings

The project seems to be two years stagnant, but it does the job well. An example use:

```
    #include <iostream>
    #include <boost/numeric/bindings/traits/ublas_vect
    #include <boost/numeric/bindings/traits/ublas_spar
    #include <boost/numeric/bindings/umfpack/umfpack.h
    #include <boost/numeric/ublas/io.hpp>

    namespace ublas = boost::numeric::ublas;
    namespace umf = boost::numeric::bindings::umfpack;

    int main() {

      ublas::compressed_matrix<double, ublas::column_m
       ublas::unbounded_array<int>, ublas::unbounded_a
(5,5,12);
      ublas::vector<double> B (5), X (5);

      A(0,0) = 2.; A(0,1) = 3;
      A(1,0) = 3.; A(1,2) = 4.; A(1,4) = 6;
      A(2,1) = -1.; A(2,2) = -3.; A(2,3) = 2.;
```

```
        A(3,2) = 1.;
        A(4,1) = 4.; A(4,2) = 2.; A(4,4) = 1.;

        B(0) = 8.; B(1) = 45.; B(2) = -3.; B(3) = 3.; B(

        umf::symbolic_type<double> Symbolic;
        umf::numeric_type<double> Numeric;

        umf::symbolic (A, Symbolic);
        umf::numeric (A, Symbolic, Numeric);
        umf::solve (A, X, B, Numeric);

        std::cout << X << std::endl;  // output: [5](1,2
    }
```

NOTE:

Though this work, I am considering moving to NETLIB

Share Improve this answer Follow

edited Nov 12, 2010 at 12:05

answered Oct 21, 2010 at 19:50

ccook
5,939 ● 6 ● 57 ● 82

This is what I ended up using. – ccook Nov 12, 2010 at 12:04