



Computer Integrated Surgery Lab
NSF Engineering Research Center for Computer Integrated
Surgical Systems and Technology

The Johns Hopkins University; Baltimore, Maryland, USA
Massachusetts Institute of Technology; Cambridge, Mass.
Brigham & Women's Hospital; Boston, Mass.
Carnegie-Mellon University; Pittsburgh, Pennsylvania
Shadyside Hospital; Pittsburgh, Pennsylvania

A Crash Course in the use of the Numerical Library for simple least squares problems

$$\begin{bmatrix} P_{N,0}(v_0) & \cdots & P_{N,N}(v_0) \\ \vdots & \ddots & \vdots \\ P_{N,0}(v_m) & \cdots & P_{N,N}(v_m) \end{bmatrix} \begin{bmatrix} c_0 \\ \vdots \\ c_m \end{bmatrix} \cong \begin{bmatrix} y_0 \\ \vdots \\ y_m \end{bmatrix}$$

A Crash Course in the use of the Numerical Library for simple least squares problems

Ver 1.0

Document CIS0.1

Date 12/01/98

Dr. Russell H Taylor

Director, Center for Computer-Integrated Surgical Systems and Technology

Room 315, New Engineering Building

3400, North Charles Street

Baltimore MD 21218

USA

Web: <http://cisstweb.cs.jhu.edu>

Phone: +01 (410) 516 0740

Fax: +01 (410) 516 5553

Email: rht@cs.jhu.edu

Developmental Software—not certified for medical use. The CISST ERC makes no warranty that the software or other information described in this document is fit for any particular purpose. Users assume full responsibility for any consequences arising from its use.

© 1998,1999 CISST ERC

This document contains proprietary and confidential information of the Engineering Research Center for Computer-Integrated Surgical Systems and Technology (CISST ERC). The contents of this document may not be disclosed to third parties, translated, copied, or duplicated in any form without the express written permission of the CISST ERC Director or other duly authorized representative of the CISST ERC. Queries should be directed to the CISST ERC Director.

1 Introduction

This document is intended to be a very short reference for one key use of the numerical library. It is not by any means a comprehensive guide to every nuance of the library's functionality.

The CIS Numerical library is a templated library of numerical routines built on top of the CIS matrix-vector library. Typical routines are declared

```
template <class scalar> routine (...)
```

where <scalar> is either "float" or "double". So far, the "double" version has been the one most used. It is believed that the "float" version will perform comparably, but there is less experience with it. For terseness in the description below, we will use "double", "doubleVector", and "doubleMatrix". It should be noted that there is an extensive library of linear algebra/numerical routines, including basic orthogonal transformations (Householder, Givens), Singular Value and QR decompositions, simple least squares solvers, and a fairly complete implementation of constrained least squares. These routines are largely based on the discussion in Lawson and Hanson's book Solving Least Squares Problems. Only a few routines are described in this very abbreviated users' guide.

Note that in this discussion we will often use A' to refer to the transpose of A . This convention is very convenient in commenting program code.

2 Singular Value Decomposition Routines

2.1 SVDRS -- used in solving $A*x=B$

// Specific headers for doubles

```
Int AlgorithmSVDRS(int mm, int nn,
                  doubleMatrix& A, // left hand side matrix (overwritten with V)
                  doubleMatrix& B, // right hand side matrix (becomes U'B)
                  doubleVector& s, // set to singular values
                  doubleVector& above, // working storage
                  doubleVector& row_h, // working storage
                  doubleVector& temp); // working storage
```

// More general templated form

```
template <class scalar>
int AlgorithmSVDRS(int mm, int nn,
                  BasicMatrix<scalar>& A,
                  BasicMatrix<scalar>& B,
                  BasicVector<scalar>& s,
                  BasicVector<scalar>& above,
                  BasicVector<scalar>& row_h,
                  BasicVector<scalar>& temp);
```

// From the subroutine header documentation (explains parameters)

```
// AlgorithmSVDRS -- This is the L&H Algorithm SVDRS
//
// Performs Singular Value Decomposition
// A = USV'
// for an mxn matrix A and returns V,S,U'B for arbitrary RHS B
// Thus U' can be computed by supplying B = I
//
// Parameters
// A = mA x n matrix, where mA>=Max(m,n)
// On Entry: Upper Left m x n submatrix contains input matrix for SVD
// On Output: Upper left n x n submatrix contains orthog matrix V
// B = m x nB matrix (ignored if nB=0)
// On Entry: Matrix of "right hand sides"
// On Exit: U'B
// s = nS vector to contain the singular values, where nS = (if m>n) then n+1 else n
// On Exit: Positive, sorted in non-increasing order
// above = nS-vector of working storage
// row_h = n-vector of working storage
// temp = max(m,n) vector of working storage
//
//Returns: ns = number of non-zero singular values
// : ns-(nS+1) if fails to converge in 10n sweeps
```

2.2 SVD – calls SVDRS

```
// AlgorithmsSVD
//
// Performs Singular Value Decomposition
// A = USV'
//
// Parameters
// A = m x n matrix
// U = m x m matrix or ignored (if U.Cols()==0)
// S = nS vector to contain the singular values, where nS = (if m>n) then
n+1 else n
// On Exit: Positive, sorted in non-increasing order
// V = n x n matrix
// On Exit: Orthog matrix V
//Returns: ns = number of non-zero singular values
//      : ns-(nS+1) if fails to converge in 10n sweeps

template<class scalar>
int AlgorithmsSVD(const BasicMatrix<scalar> A,
                 BasicMatrix<scalar>& Ut,
                 BasicVector<scalar>& S,
                 BasicMatrix<scalar>& V)
;
```

3

4 Coding Examples

4.1 A simple numerical example – all you need to do least squares

```
#include <numerical.h>

//
// Computer Integrated Surgery Lab
// The Johns Hopkins University
// Copyright (C) 1999
// Do not copy or release without
// written permission from Russell H. Taylor, CIS Lab Director
//
// Author: Russell H. Taylor
//   rht@cs.jhu.edu
//

#include <numerical.h>

/// This subroutine solves least squares problems AX=B for doubleMatrix and floatMatrix
/// Example of use is below

template <class scalar>
  int SVDLeastSquares(const BasicMatrix<scalar>& A,
                    const BasicMatrix<scalar>& B,
                    BasicMatrix<scalar>& X)
{ int mA = A.Rows();
  int nA = A.Cols();
  int mB = B.Rows();
  int nB = B.Cols();
  MatrixVectorSubscriptAssert(nA==X.Rows(),B); // some standard bug trapping code
  MatrixVectorSubscriptAssert(mA==mB&&nB==X.Cols(),X); // ditto

  MatrixVectorSubscriptAssert(mA>=nA,A); // this routine only works in this case

  doubleMatrix AV = A;
  doubleMatrix V(MakeReference,AV,SubscriptRange(0,nA-1),SubscriptRange(0,nA-1));
  doubleMatrix UtB = B;

  doubleVector tempS(nA+1);
  doubleVector above(nA+1);
  doubleVector row_h(nA);
  doubleVector temp(mA);

  int nS = AlgorithmSVDRS(mA,nA,AV,UtB,tempS,above,row_h,temp);

  assert(nS==nA); // really should include better error check here
                 // this code is assuming A is full rank, just for
                 // simplicity

  BasicVector<scalar> Y(nS);

  for (int k=0;k<nB;k++)
```

```

    {
        for (int i=0;i<nS;i++)
            { Y(i) = UtB(i,k)*(1.0/tempS(i));};
        Inner(V,Y,X(k));
    };

return nS;

};

////////// Example of use

#include <RandomValues.h> // just generate random data
#include <RandomMatrices.h>

void SimpleLeastSquaresTest()
{
    IndentedOstream IOS = cout;

    doubleMatrix A(10,5);
    doubleMatrix B(10,1);
    doubleMatrix X(5,1);
    doubleMatrix AX(10,1);
    doubleMatrix E(10,1);

    RandomFill(A,-10.,10.);
    RandomFill(X,-10.,10.);
    RandomFill(E,-.1,.1);
    AX = A*X;

    B=AX+E;

    SVDLeastSquares(A,B,X); // Solves the problem

    E=AX-B;

    IOS << E<<endl;

};

```

5 Contact Information

Document Maintainer Russ Taylor (rht@cs.jhu.edu)

Software Maintainer Russ Taylor (rht@cs.jhu.edu)

Repository CIS Lab, JHU

Authors Russ Taylor (rht@cs.jhu.edu)

Acknowledgments