

The IMSL[®] C Library Error Handler

IMSL Technical Report P10550

A White Paper by Visual Numerics, Inc.

November 2008

Visual Numerics[®]

Visual Numerics, Inc.
2500 Wilcrest Drive, Suite 200
Houston, TX 77042
USA

www.vni.com

The IMSL C Library Error Handler

IMSL Technical Report P10550

by **Visual Numerics, Inc.**

© 2008 by Visual Numerics, Inc. All Rights Reserved
Printed in the United States of America

Publishing History:

November 2008

Trademark Information

Visual Numerics, IMSL and PV-WAVE are registered trademarks. JMSL TS-WAVE, JWAVE and PyIMSL are trademarks of Visual Numerics, Inc., in the U.S. and other countries. All other product and company names are trademarks or registered trademarks of their respective owners.

The information contained in this document is subject to change without notice. Visual Numerics, Inc. makes no warranty of any kind with regard to this material, included, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Visual Numerics, Inc, shall not be liable for errors contained herein or for incidental, consequential, or other indirect damages in connection with the furnishing, performance, or use of this material.

TABLE OF CONTENTS

Audience	4
Introduction	4
What Determines Error Severity.....	4
Kinds of Errors and Default Actions	4
Examples	6
Example 1: Disable Default Stopping and Printing	6
Example 2: C++ Exception.....	8
Example 3: Custom Error Printing Function	11
About the Author	12

Audience

This paper is intended for developers creating applications that include calls to the IMSL C Numerical Library. IMSL C Library functions and capabilities for error handling are discussed and example programs are provided.

This technical note is written for the IMSL C Library – Math Library. The same concepts apply for the IMSL Stat Library.

Introduction

IMSL C Library functions attempt to detect user errors and handle them in a way that provides as much information to the user as possible. To do this, the IMSL C Library recognizes various levels of severity of errors, and also considers the extent of the error in the context of the purpose of the function; a trivial error in one situation may be serious in another. An error code is also associated with each error issued from the IMSL C Library. Error codes, together with the error severity, can be useful in determining action after returning from an IMSL C Library function.

What Determines Error Severity

In some cases, the user's input may be mathematically correct, but because of limitations of the computer arithmetic and of the algorithm used, it is not possible to compute an answer accurately. In this case, the assessed degree of accuracy determines the severity of the error. In cases where the function computes several output quantities, if some are not computable but most are, an error condition exists; and its severity depends on an assessment of the overall impact of the error.

Kinds of Errors and Default Actions

Five levels of severity of errors are defined in the IMSL C Math Library. Each level has an associated PRINT attribute and a STOP attribute. These attributes have default settings (YES or NO), but they may also be set by the user. The purpose of having multiple error types is to provide independent control of actions to be taken for errors of different levels of severity. Upon return from a C Math Library function, exactly one error state exists. (A code 0 "error" is no error.) Even if more than one informational error occurs, only one message is printed (if the PRINT attribute is YES). Multiple errors for which no corrective action within the calling program is reasonable or necessary result in the printing of multiple messages (if the PRINT attribute for their severity level is YES). Errors of any of the severity levels, except IMSL_TERMINAL, may be informational errors. The include file, `imsl.h` (`imsls.h` for the C Stat Library), defines `IMSL_NOTE`, `IMSL_ALERT`, `IMSL_WARNING`, `IMSL_FATAL`, `IMSL_TERMINAL`, `IMSL_WARNING_IMMEDIATE`, and `IMSL_FATAL_IMMEDIATE` as an enumerated data type `Imsl_error`.

IMSL_NOTE. A note is issued to indicate the possibility of a trivial error or simply to provide information about the computations.

Default attributes: PRINT=NO, STOP=NO.

IMSL_ALERT. An alert indicates that a function value has been set to 0 due to underflow.

Default attributes: PRINT=NO, STOP=NO.

IMSL_WARNING. A warning indicates the existence of a condition that may require corrective action by the user or calling routine. A warning error may be issued because the results are accurate to only a few decimal places, because some of the output may be erroneous, but most of the output is correct, or because some assumptions underlying the analysis technique are violated. Usually no corrective action is necessary, and the condition can be ignored.

Default attributes: PRINT=YES, STOP=NO.

IMSL_FATAL. A fatal error indicates the existence of a condition that may be serious. In most cases, the user or calling routine must take corrective action to recover.

Default attributes: PRINT=YES, STOP=YES.

IMSL_TERMINAL. A terminal error is serious. It usually is the result of an incorrect specification, such as specifying a negative number as the number of equations. These errors may also be caused by various programming errors. The resulting error message may be perplexing to the user. In such cases, the user is advised to compare carefully the actual arguments passed to the function with the argument descriptions given in the documentation. Special attention should be given to checking argument order and data types.

A terminal error is not an informational error, because corrective action within the program is generally not reasonable. In normal usage, execution is terminated immediately when a terminal error occurs. Messages relating to more than one terminal error are printed if they occur.

Default attributes: PRINT=YES, STOP=YES.

IMSL_WARNING_IMMEDIATE. An immediate warning error is identical to a warning error, except it is printed immediately.

Default attributes: PRINT=YES, STOP=NO.

IMSL_FATAL_IMMEDIATE. An immediate fatal error is identical to a fatal error, except it is printed immediately.

Default attributes: PRINT=YES, STOP=YES.

The user can set PRINT and STOP attributes by calling `imsl_error_options` as described in Chapter 12, “Utilities” of the C Math Library User Guide.

Functions for Error Handling

There are two ways in which the user may interact with the error handling system:

1. to change the default actions
2. to determine the elements (error code, error severity and error message) of an error

The relevant functions are:

Function	IMSL C Library version that introduced the function
<code>imsl_error_options</code>	v1.0
<code>imsl_error_code</code>	v1.0
<code>imsl_error_type</code>	v7.0
<code>imsl_error_message</code>	v7.0

Function `imsl_error_options` sets the actions to be taken when errors occur. Functions `imsl_error_code`, `imsl_error_type`, and `imsl_error_message` retrieve code, severity, and message text for an informational error.

Examples

Example 1: Disable Default Stopping and Printing

This example calls `imsl_f_lin_sol_gen` to compute the inverse of an ill-conditioned matrix, which could result in an error being issued from C Math Library. However, because stopping and printing of errors is disabled, C Math Library error message(s) will not be displayed and execution will not be halted, allowing the program to take corrective action. The corrective action is to reduce the size of the matrix.

```
#include <imsl.h>
#include <stdio.h>

#define NMAX 10 /* Maximum matrix order for this example. */

int main(void)
{
    char    *msg=NULL;
    int     i, j, n, type=0, code=0;
    float   a[NMAX*NMAX], *b=NULL, *inva;

    printf("\nInverting successively smaller Hilbert matrices, ");
    printf("starting with size %d x %d.\n", NMAX, NMAX);
    printf("If an error occurs, reduce the order of the matrix and try \n");
    printf("again until the matrix is inverted without an error being ");
    printf("issued.\n\n");

    /*
     * Disable printing and stopping on all C/Math/Library errors.
     */
    imsl_error_options(IMSL_SET_PRINT, IMSL_NOTE, 0,
```

```

        IMSL_SET_PRINT, IMSL_ALERT, 0,
        IMSL_SET_PRINT, IMSL_WARNING, 0,
        IMSL_SET_PRINT, IMSL_FATAL, 0,
        IMSL_SET_PRINT, IMSL_TERMINAL, 0,
        IMSL_SET_PRINT, IMSL_WARNING_IMMEDIATE, 0,
        IMSL_SET_PRINT, IMSL_FATAL_IMMEDIATE, 0,
        IMSL_SET_STOP, IMSL_NOTE, 0,
        IMSL_SET_STOP, IMSL_ALERT, 0,
        IMSL_SET_STOP, IMSL_WARNING, 0,
        IMSL_SET_STOP, IMSL_FATAL, 0,
        IMSL_SET_STOP, IMSL_TERMINAL, 0,
        IMSL_SET_STOP, IMSL_WARNING_IMMEDIATE, 0,
        IMSL_SET_STOP, IMSL_FATAL_IMMEDIATE, 0,
    0);

for (n=NMAX; n>1; n--) {
    /*
     * Fill a[] with a (n x n) Hilbert matrix.
     */
    for (i=0;i<n;i++)
        for (j=0;j<n;j++)
            a[i*n+j] = 1.0/(i+j+1);

    /*
     * Call imsl_f_lin_sol_gen to compute the inverse.
     */
    imsl_f_lin_sol_gen(n, a, b,
                      IMSL_INVERSE, &inva,
                      IMSL_INVERSE_ONLY,
                      0);

    /*
     * Check if an error occurred in imsl_f_lin_sol_gen.
     * If an error occurred:
     * - record the type, code and message
     * - return to top of for-loop to reduce the order of the matrix
     *   and try again.
     * If an error did not occur, then break out of the for-loop.
     */
    if (imsl_error_code() != 0) {
        /* Record the type, code and text of the error. */
        type = imsl_error_type();
        code = imsl_error_code();
        if (msg != NULL) imsl_free(msg);
        msg = imsl_error_message();
    } else {
        break;
    }
}
/*
 * Output results.
 */
if (code != 0) {
    printf("The largest Hilbert matrix inverted without an error\n");
    printf("being issued was %d x %d.\n\n", n, n);
    printf("The error issued for the matrix of size %d x %d was:\n",
           n+1, n+1);
    printf("\t\ttype = %d\n\tcode = %d\n", type, code);
}

```

```

        printf("\tmsg = %s\n", msg);
    } else {
        printf("No errors were issued by imsl_f_lin_sol_gen.\n\n");
    }
}

```

Output

Inverting successively smaller Hilbert matrices, starting with size 10 x 10. If an error occurs, reduce the order of the matrix and try again until the matrix is inverted without an error being issued.

The largest Hilbert matrix inverted without an error being issued was 5 x 5.

The error issued for the matrix of size 6 x 6 was:

```

    type = 3
    code = 1003
    msg = The matrix is too ill-conditioned.  An estimate of the
reciprocal of its L1 condition number is "rcond" = 3.540551e-08. The solution
might not be accurate.

```

Example 2: C++ Exception

C Math Library functions are often called from applications that rely on C++ exceptions. This example demonstrates a simple way to customize the C Math Library error handling capabilities to allow for C++ exceptions to be thrown by the application in case of serious errors from C Math Library.

In this example, the C++ exception class `IMSLCMathExc` extends the standard C++ exception class. All printing and stopping on C Math Library errors are disabled and a check for errors is made upon returning from C Math Library functions. This example attempts to compute the inverse of three different matrices. The first matrix is inverted without error, the second results in a warning error from C Math Library, and the third results in a fatal error from C Math Library. All errors are logged, and a C++ exception is thrown in the case of fatal or terminal errors.

```

#include <iostream>
#include <exception>
#include <imsl.h>
using namespace std;

void inv(int n, float *a);
void silence_cmath_errors();
void check_for_cmath_error(const char* fcn);
void log_cmath_error(const char* fcn);

class IMSLCMathException: public exception
{
    virtual const char* what() const throw()
    {
        return "### IMSLCMathException\n" //

```

```

        "### A Fatal or Terminal C/Math error has occurred. \n";
    }
} IMSLMathExc;

int main(void)
{
    // Matrix a1 is non-singular.
    float  a1[] = {1.0, 3.0, 3.0,
                  1.0, 3.0, 4.0,
                  1.0, 4.0, 3.0};

    // Matrix a2 will be filled with an ill-conditioned matrix.
    float  a2[6*6];

    // Matrix a3 is singular.
    float  a3[] = {0.0, 3.0, 3.0,
                  0.0, 3.0, 4.0,
                  0.0, 4.0, 3.0};

    // Fill a2[] with a (6 x 6) ill-conditioned (Hilbert) matrix.
    for (int i=0;i<6;i++)
        for (int j=0;j<6;j++)
            a2[i*6+j] = 1.0/(i+j+1);

    // Disable printing and stopping on all C/Math/Library errors.
    silence_cmath_errors();

    // Compute the inverse of a1, a2, and a3 using IMSL C/Math.
    try
    {
        // Inverse of a1 should be computed without error.
        cout << "\n...computing the inverse of a1" << endl;
        inv(3, a1);

        // A warning is issued when inverting this matrix.
        cout << "\n...computing the inverse of a2" << endl;
        inv(6, a2);

        // Matrix a3 is singular.  Expect an exception to be thrown.
        cout << "\n...computing the inverse of a3" << endl;
        inv(3, a3);
    }
    catch (exception& e)
    {
        cout << e.what();
    }
}

// Inverts an n by n matrix a using IMSL C/Math.
void inv(int n, float *a)
{
    float *b = 0, *inva;
    imsl_f_lin_sol_gen(n, a, b,
                      IMSL_INVERSE, &inva,
                      IMSL_INVERSE_ONLY,
                      0);
}

```

```

    check_for_cmath_error("imsl_f_lin_sol_gen() inside inv()");

    imsl_f_write_matrix((char*)"Computed Inverse", n, n, inva, 0);
    check_for_cmath_error("imsl_f_write_matrix() inside inv()");

    imsl_free(inva);
}

// Check if an error has been issued by C/Math.
// All errors are logged. FATAL and TERMINAL errors result in
// an exception being thrown.
void check_for_cmath_error(const char *fcn)
{
    if (imsl_error_code()) {
        log_cmath_error(fcn);
        switch(imsl_error_type()) {
            case IMSL_FATAL:
            case IMSL_TERMINAL:
            case IMSL_FATAL_IMMEDIATE:
                throw IMSLCMathExc;
                break;
            default:
                break;
        }
    }
}

// Log details of errors from IMSL C/Math.
void log_cmath_error(const char *fcn)
{
    char *types[] = {" ", "Note", "Alert", "Warning", "Fatal", "Terminal",
                    "Warning Immediate", "Fatal Immediate"};
    char *msg = imsl_error_message();

    cout << "*** A C/Math Error occurred when calling "<< fcn << endl;
    cout << "    Type: " << types[imsl_error_type()] << endl;
    cout << "    Code: " << imsl_error_code() << endl;
    cout << "    " << msg << endl;

    imsl_free(msg);
}

// Disable printing and stopping for all C/Math errors.
void silence_cmath_errors()
{
    imsl_error_options(IMSL_SET_PRINT, IMSL_NOTE, 0,
                      IMSL_SET_PRINT, IMSL_ALERT, 0,
                      IMSL_SET_PRINT, IMSL_WARNING, 0,
                      IMSL_SET_PRINT, IMSL_FATAL, 0,
                      IMSL_SET_PRINT, IMSL_TERMINAL, 0,
                      IMSL_SET_PRINT, IMSL_WARNING_IMMEDIATE, 0,
                      IMSL_SET_PRINT, IMSL_FATAL_IMMEDIATE, 0,
                      IMSL_SET_STOP, IMSL_NOTE, 0,
                      IMSL_SET_STOP, IMSL_ALERT, 0,
                      IMSL_SET_STOP, IMSL_WARNING, 0,
                      IMSL_SET_STOP, IMSL_FATAL, 0,
                      IMSL_SET_STOP, IMSL_TERMINAL, 0,

```

```

        IMSL_SET_STOP, IMSL_WARNING_IMMEDIATE, 0,
        IMSL_SET_STOP, IMSL_FATAL_IMMEDIATE, 0,
        0);
}

```

Output

...computing the inverse of a1

	Computed Inverse		
	1	2	3
1	7	-3	-3
2	-1	0	1
3	-1	1	0

...computing the inverse of a2

```

*** A C/Math Error occurred when calling imsl_f_lin_sol_gen() inside inv()
    Type: Warning
    Code: 1003

```

The matrix is too ill-conditioned. An estimate of the reciprocal of its L1 condition number is "rcond" = 3.540551e-08. The solution might not be accurate.

	Computed Inverse					
	1	2	3	4	5	6
1	36	-619	3285	-7364	7344	-2686
2	-619	14378	-86024	206010	-214226	80682
3	3282	-85995	549575	-1372375	1469054	-565160
4	-7355	205888	-1372056	3526863	-3856269	1507808
5	7332	-214056	1468463	-3855647	4284664	-1696877
6	-2682	80605	-564860	1507388	-1696683	678933

...computing the inverse of a3

```

*** A C/Math Error occurred when calling imsl_f_lin_sol_gen() inside inv()
    Type: Fatal
    Code: 1004

```

The input matrix is singular.

```

### IMSLMathException

```

```

### A Fatal or Terminal C/Math error has occurred.

```

Example 3: Custom Error Printing Function

Supplying a custom error printing function can be useful in a number of situations, such as:

- Application-specific formatting of error messages.
- Logging errors in multiple files.
- Issuing error messages in application-specific dialog boxes.

```

#include <imsl.h>
#include <stdio.h>

void custom_print_proc(Imsl_error, long, char*, char*);

int main(void)
{

```

```

/*
 * Supply a custom error print function to C/Math/Library.
 */
imsl_error_options(IMSL_ERROR_PRINT_PROC, custom_print_proc, 0);

/*
 * Call imsl_f_gamma() with a value that will cause an error.
 */
imsl_f_gamma(100.0);
}

/*
 * Custom error printing function.
 */
void custom_print_proc(Imssl_error type, long code, char *function_name,
                      char *message)
{
    char *type_names[] = {" ", "Note", "Alert", "Warning", "Fatal",
                          "Terminal", "Warning Immediate",
                          "Fatal Immediate"};

    printf("\n%s error from IMSL function %s(): %s\n\n",
          type_names[type], function_name, message);
}

```

Output

```

Fatal error from IMSL function imsl_f_gamma(): The function overflows because
"x" = 1.000000e+02 is greater than 3.503078e+01.

```

About the Author

Michael (Mike) Pulverenti is a Technical Program Manager at Visual Numerics, currently responsible for managing the technical aspects of relationships with OEM partners and developers of Visual Numerics Affiliated Products. In previous roles at the company, Mike has been involved in all aspects of the software development process for the IMSL Numerical Libraries, including development, testing, porting and maintaining of numerical algorithms in C#, C, Fortran and Java.