IMSL C Numerical Library 6.0
User Training

Visual Numerics, Inc. Greater China
大中華區技術部經理
劉泰興 Ted Liu
ted@vni.com.tw

Schedule & Rules

- Session starts at 13:00
- Lecture / Break / Lecture
- Class end around 16:30
- Practice from 16:30 to 17:30
- This class is informal
  - Please ask questions, make comments, discussion, etc.

Agenda

- What's New in IMSL CNL 6.0
- IMSL CNL Getting Started
- How to Compile and Link IMSL CNL
- IMSL CNL Inside
- Open Discussion

Visual Numerics

www.vni.com.tw

About Visual Numerics

Company

- Leader in advanced numerical analysis and visualization software
- Established 1970; privately held
- Direct sales, support, and distributor channels worldwide

Products & Services

- Embeddable mathematical and statistical algorithms for a broad range of applications
- Visualization software for data-intensive, production applications
- Consultants with unique expertise in computational science and algorithm development

Customers & Partners

- Over 500,000 users worldwide
- Core markets in high performance computing (HPC), finance and business analytics
- Long-term relationships with top tier technology partners

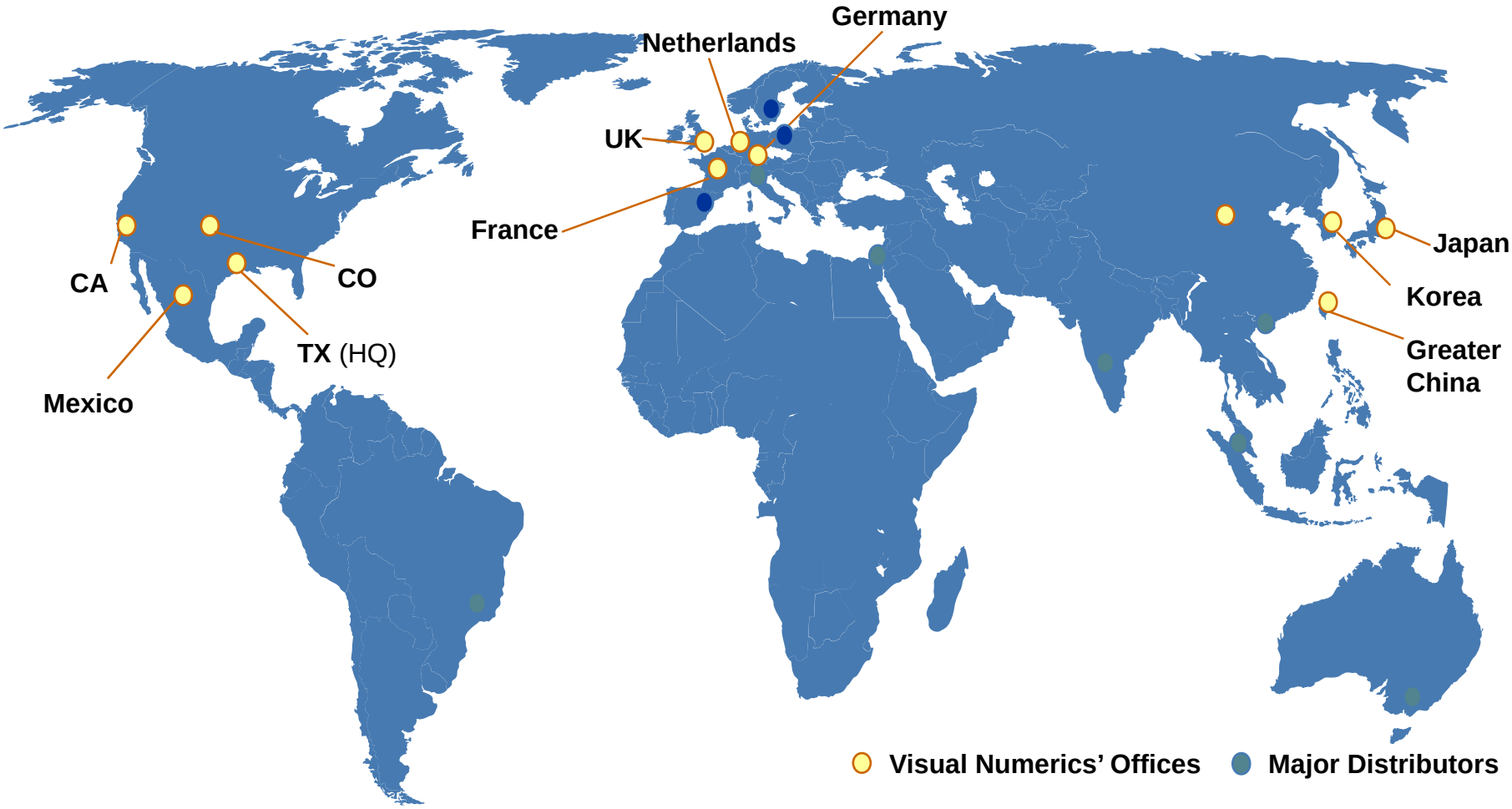
Visual Numerics, Inc. History

- 1970
  - IMSL, Inc. (Texas, USA)
- 1980
  - Precision Visuals, Inc. (Colorado, USA)
- 1993
  - IMSL, Inc. and Precision Visuals, Inc. merged to Visual Numerics, Inc.
- 1995
  - Visual Numerics, Inc. Greater China established
- 2007
  - 37th Anniversary

Visual Numerics

www.vni.com.tw

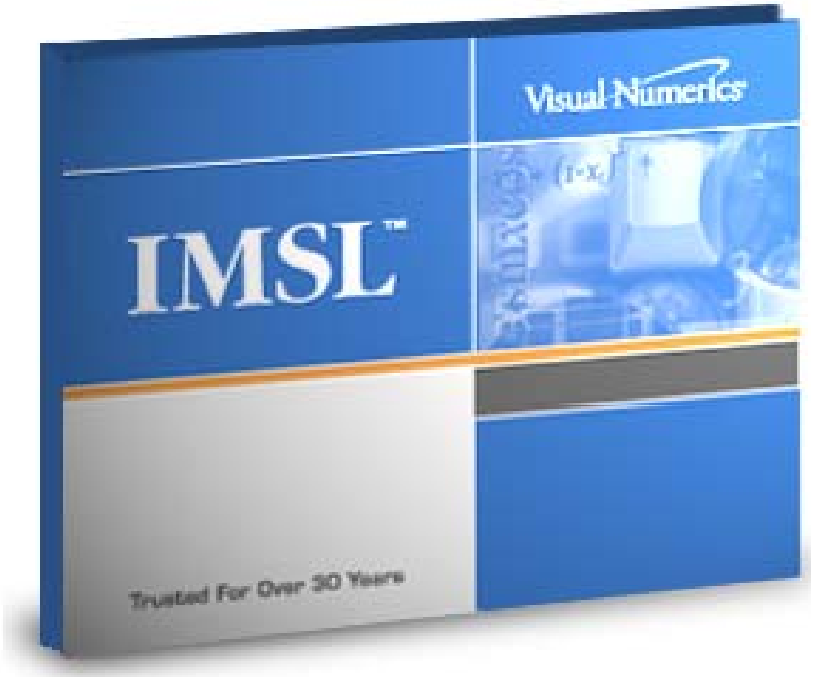
Visual Numerics Global Footprint



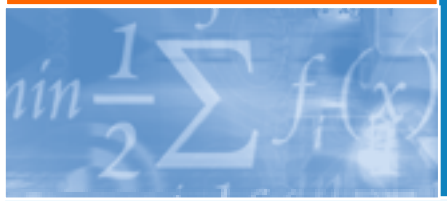
Visual Numerics

www.vni.com.tw

IMSL™ Numerical Libraries for C, C#, Java™, and Fortran



- Comprehensive libraries of mathematical, statistical, and financial numerical algorithms
- Embeddable code eliminates the need to write code from scratch
- High-performance computing and expertise for developing and executing sophisticated numerical applications
- Trusted for over 30 years



New in IMSL C Numerical Library 6.0

New for IMSL CNL 6.0 Linear Programming

- World's fastest dense Linear Programming Optimizer in a general math library
- World's most robust dense Linear Programming Optimizer in a general math library
- MPS reader
  - Common file format for optimization problems

Visual Numerics

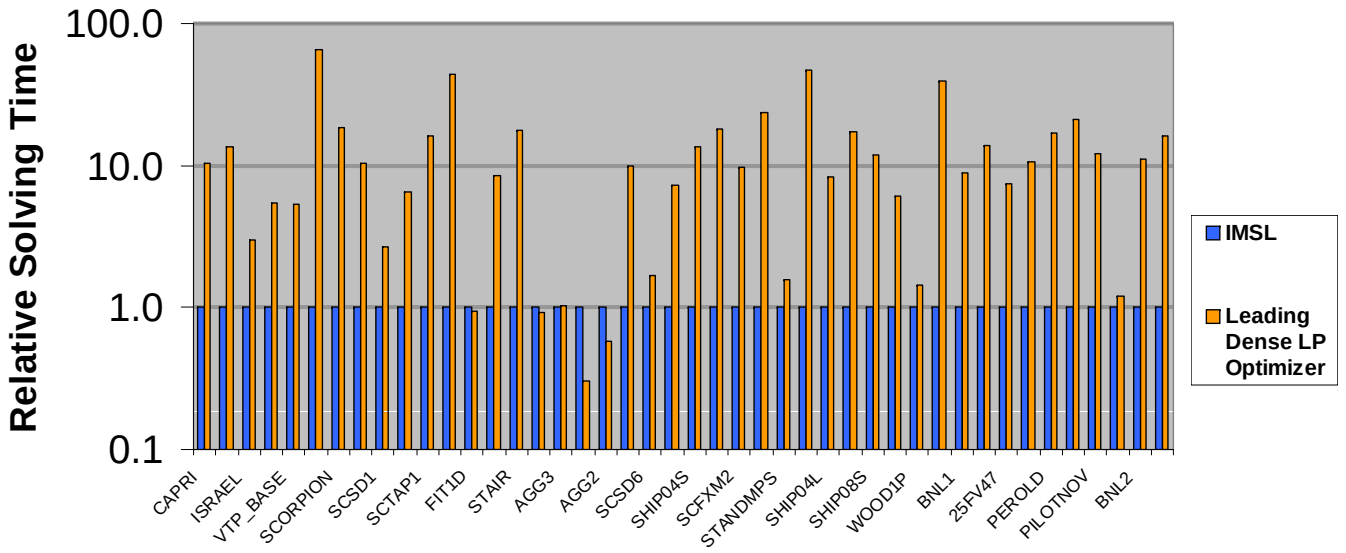
New for IMSL CNL 6.0
Linear Programming Speed

- Extensive testing on a suite of test cases from Netlib (http://www-fp.mcs.anl.gov/otc/Guide/TestProblems/LPtest/)

Dense Linear Programming Problems Relative Solving Time

Sample Problems from Netlib library

44 Example problems that both IMSL and a leading dense LP Optimizer could solve and for which timing could be measured

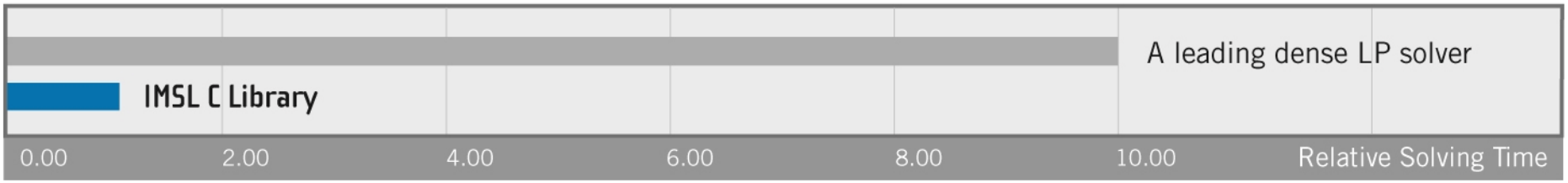


Visual Numerics

New for IMSL CNL 6.0
Linear Programming Speed

- Over 10 times faster than a leading dense Linear Programming Optimizer

Average Solving Time For IMSL Versus a Competitve Dense LP Solver (44 Measurable Example Problems)



IMSL C Library solves linear programming problems in 1/10 the time.

New for IMSL CNL 6.0
Linear Programming Robustness

- 91 problems attempted by IMSL and another LP Optimizer
  - IMSL solved 91, failed on 0
  - Other dense LP Optimizer solved 56, failed on 35
- IMSL provides clear diagnostics if problem is infeasible
  - A Leading dense LP Optimizer keeps plugging away without indication of status
  - Can take more hours until user finally gives up

Problems Solved for Dense LP Solvers

IMSL C Library solves 100% of sample Netlib problems.

New for IMSL CNL 6.0 Forecasting Package

- Neural Network
  - Data pre-conditioners
  - Trainer
  - Forecasting engine
- AutoARIMA forecasting algorithm
  - Automatically identifies outliers, classifies outliers, and estimates missing values
- Time-series data pre-conditioners
  - Outlier detection
  - Outlier classification
  - Missing value estimation
- Stand-alone pre-conditioners

New for IMSL CNL 6.0
Mersenne Twister / PDE

- Mersenne Twister
  - Fast, efficient
  - Very high-quality random numbers
  - Long Period
  - Chance of any detectable pattern is reduced
  - Developed 1996-1997 in Japan (Makoto Matsumoto & Takuji Nishimura)
- Partial Differential Equations
  - dea_petzold_gear
    - Solution of Differential–Algebraic Systems using Petzold-Gear method
  - pde_1d_mg
    - Solves a system of one-dimensional time-dependent partial differential equations using a moving-grid interface

Visual Numerics

www.vni.com.tw

IMSL CNL 6.0 Ports on Windows & Linux

Operation System	Compiler Version
Windows 32-bit (x86-32)	**MSVS 2005** **MSVS .NET 2003** **MSVC++ 6.0** **Intel 9.0**
Windows 64-bit (x86-64)	**Opteron Intel 9.0** **Opteron MS SDK** **EM64T Intel 9.0** **EM64T MS SDK**
Linux 32-bit (x86-32)	**RH4 gcc 3.4.3** **RH4 Intel 9.1** **Mac OSX 10.4.8 gcc 4.0.1**
Linux 64-bit (x86-64)	**Opteron RH 4 gcc3.4.3** **Opteron SuSE 9 gcc3.3.3 (SuSE10 gcc 4.1.0 OK)** **Opteron SuSE 9 Portland pgcc 6.0-5** **EM64T SuSE 9 3.3.3** **EM64T RH 4 gcc3.4.4** **EM64T RH 4 Intel 9.1**
Linux 64-bit (Itanium2)	**Red Hat EL 4.0 Intel 9.0** **SuSE 9 Intel 9.0** **VMS 8.3 C 7.1.011**

New for IMSL CNL 6.0 LAPACK Integration

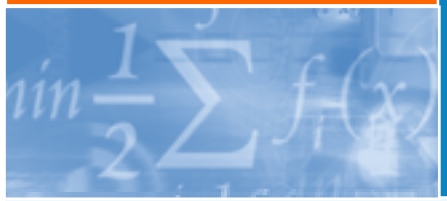
- Many of the codes in the IMSL C Numerical Library are based on **LINPACK** and **EISPACK**
  - Designed in the 1970s and early 1980s
- LAPACK (1999) http://www.netlib.org/lapack was designed to make the linear solvers and eigensystem routines run more efficiently on high performance computers
- IMSL C Library has the option of linking to code which is based on either the legacy routines or the more efficient LAPACK routines

New for IMSL CNL 6.0
LAPACK Integration

Single Precision IMSL Library Routine	LAPACK Routines used when Linking with Name
imsl_c_eig_gen	?geevx_, ?=c/z
imsl_c_eig_herm	?heevx_, ?=c/z
imsl_c_geneig	?ggev_, ?=c/z
imsl_c_lin_sol_gen	?getrs_, ?getrf_, ?gecon_, ?getri_, ?=c/z
imsl_c_lin_sol_gen_band	?gbcon_, ?gbtrf_, ?gbtrs_, ?=c/z
imsl_c_lin_sol_posdef	?potri_, ?potrf_, ?pocon_, ?potrs_, ?=c/z
imsl_c_lin_sol_posdef_band	?pbcon_, ?pbtrf_, ?pbtrs_, ?=c/z
imsl_c_lin_svd_gen	?gesvd_, ?=c/z
imsl_f_bounded_least_squares	?ormqr_, ?=s/d
imsl_f_eig_gen	?geevx_, ?=s/d
imsl_f_eig_sym	?yevx_, ?=s/d
imsl_f_eig_symgen	?sygv_, ?=s/d
imsl_f_geneig	?ggev_, ?=s/d
imsl_f_lin_least_squares_gen	?orgqr_, ?geqrf_, ?geqp3_, ?ormqr_, ?trsm_, ?=s/d
imsl_f_lin_sol_gen	?getrs_, ?getrf_, ?gecon_, ?getri_, ?=s/d
imsl_f_lin_sol_gen_band	?gbcon_, ?gbtrf_, ?gbtrs_, ?=s/d
imsl_f_lin_sol_posdef	?pocon_, ?potrf_, ?potri_, ?=s/d
imsl_f_lin_svd_gen	?gesvd_, ?=s/d
imsl_f_nonlin_least_squares	?ormqr_, ?=s/d
imsl_f_pde_1d_mg_mgr	?gbtrf_, ?=s/d

IMSL CNL Vendor BLAS

- Basic Linear Algebra Subroutines http://www.netlib.org/blas/
  - Level 1: vector-vector operations
  - Level 2: matrix-vector operations
  - Level 3: matrix-matrix operations
- A user-supplied BLAS library is required using the SMP linking options
  - Set **OMP_NUM_THREADS** to set the number of CPUs
- Expanded support
  - IMSL has some functions that use vendor BLAS

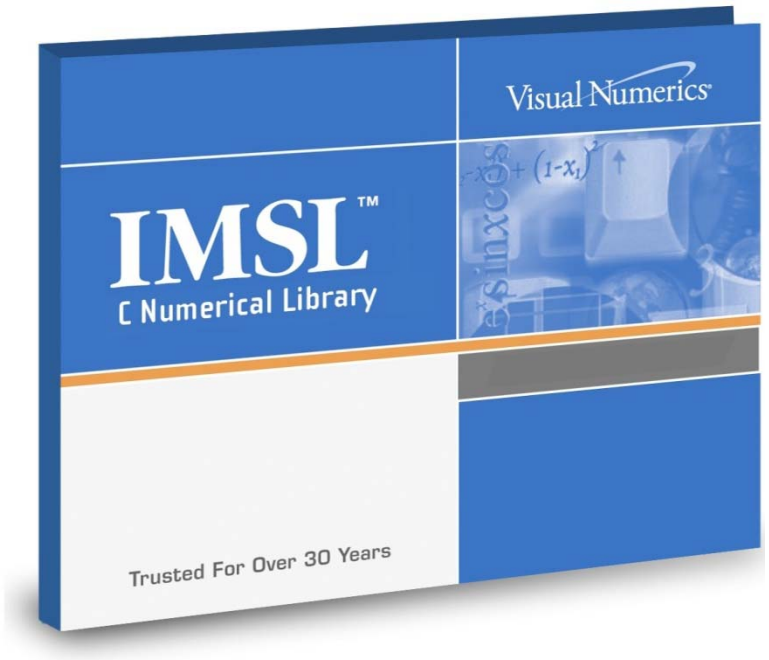


IMSL CNL Getting Started

Visual Numerics

www.vni.com.tw

IMSL C Numerical Library



- Thread-Safe
- SMP High-Performance Technology
- 100% Pure C Code
- Cost-Effectiveness and Value
- Intuitive Programming
- Diagnostic Error Handling
- Fully tested and qualified for popular hardware, operating systems, and compilers

IMSL License Types

- Development / Run-Time
- Object code / Source code
- Node-Locked / Floating
- Campus and Department Annual Licenses
- Evaluation

IMSL C Library History

Year	Major Release
1990	1st release of IMSL C library (Ver.1.0 – Math)
1991	Ver.1.0 - Stat release
1995	Ver.2.0 release (included library for C++)
1998	Ver.2.5 release
1999	Ver.3.0 release
2000	Ver.4.0 release (100% thread safe)
2001	Ver.5.0 release Kalman filter, QMC, Faure sequence, finance & bond
2003	Ver.5.5 release NLP, enhancement of time-series and multivariate analysis, DOE
2006	Ver.6.0 release Fast LP, AutoARIMA, neural network, time-series data pre-conditioner, Mersenne Twister random number, LAPACK

IMSL CNL MATH Functions

1. Linear Systems
2. Eigensystem Analysis
3. Interpolation and Approximation
4. Quadrature
5. Differential Equations
6. Transforms
7. Nonlinear Equations
8. Optimization
9. Special Functions
10. Statistics and Random Number Generation
11. Printing Functions
12. Utilities

IMSL CNL STAT Functions

1. Basic Statistics
2. Regression
3. Correlation and Covariance
4. Analysis of Variance and Designed Experiments
5. Categorical and Discrete Data Analysis
6. Nonparametric Statistics
7. Tests of Goodness-of-Fit
8. Time Series and Forecasting
9. Multivariate Analysis
10. Survival and Reliability Analysis
11. Probability Distribution Functions and Inverses
12. Random Number Generator
13. Neural Networks
14. Printing Functions
15. Utilities

IMSL CNL 6.0 Directory Layout UNIX Versus Windows

<VNI_DIR> (/opt/vni/)

- license
  - bin
  - bin.rs6000
- imsl
  - cnl600
    - help
    - **itanium**
    - bin
    - examples
    - **include**
    - **lib**
    - notes

<VNI_DIR> (C:\Program Files\VNI)

- license
  - bin
  - bin.i386nt
- imsl
  - cnl600
    - help
    - **vs05pc**
      - bin
      - examples
      - **include**
      - **lib**
      - notes

IMSL CNL Naming Conventions 1/3

- Most functions are available in both a type *float* and a type *double* version, with names of the two versions sharing a common root
- The section names for the functions only contain the common root to make finding the functions easier
- For example
  - lin_sol_gen (common root)
    - imsl_f_lin_sol_gen
    - imsl_d_lin_sol_gen

IMSL CNL Naming Conventions 2/3

- Same variable name is used consistently throughout the IMSL
- For example
  - In the functions for eigensystem analysis
    - eval
      - the vector of eigenvalues
    - n_eval
      - the number of eigenvalues computed or to be computed

IMSL CNL Naming Conventions 3/3

C Math Library	
Type	**Prefix**
float	imsl_f_
double	imsl_d_
int	imsl_i_
f_complex	imsl_c_
d_complex	imsl_z_

C Stat Library	
Type	**Prefix**
float	Imsls_f_
double	imsls_d_
int	imsls_i_

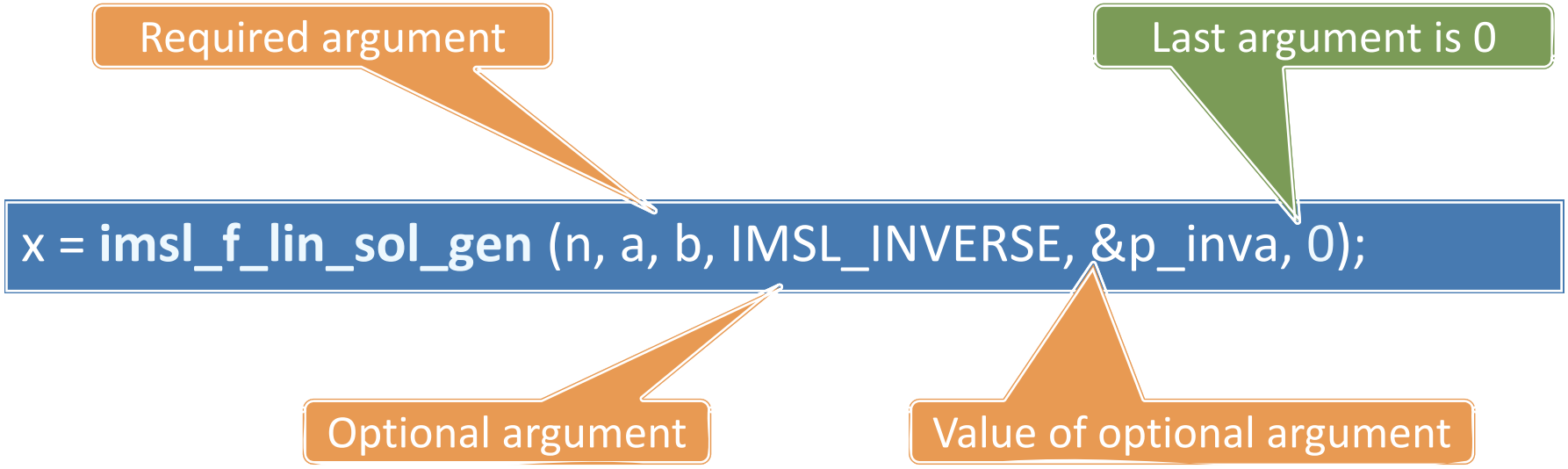
- Do not choose a name beginning with “imsls_” or “imsl_” in any combination of uppercase or lowercase characters

IMSL CNL Include File

- Located at /opt/vni/imsl/cnl600/include
- Specify at the top of the program
  - C/MATH - #include <imsl.h>
  - C/STAT - #include <imsls.h>

IMSL CNL Arguments

- Use of optional argument
- The last argument is always 0



IMSL CNL
How to Find the Right Routine

- Locate in each chapter introduction
  - Table of contents located in chapter introduction
  - Alphabetical “Summary of Functions”
- Each routine in the document has at least one example demonstrating its application

Visual Numerics

IMSL CNL Manual Organization Example: LIN_SOL_GEN 1/5

lin_sol_gen

Solves a real general system of linear equations Ax = b. Using optional arguments, any of several related computations can be performed. These extra tasks include computing the LU factorization of A using partial pivoting, computing the inverse matrix A^{-1}, solving $A^T x = b$, or computing the solution of Ax = b given the LU factorization of A.

Synopsis

#include <imsl.h>

float ***imsl_f_lin_sol_gen** (int n, float a[], float b[], …, 0)

The type double procedure is **imsl_d_lin_sol_gen**.

Required Arguments

int n (Input)

Number of rows and columns in the matrix.

float a[] (Input)

Array of size *n* × *n* containing the matrix.

float b[] (Input)

Array of size *n* containing the right-hand side.

IMSL CNL Manual Organization Example: LIN_SOL_GEN 2/5

Return Value

A pointer to the solution *x of the linear system Ax = b. To release this space, use free*. If no solution was computed, then NULL is returned.

Synopsis with Optional Arguments

```
#include <imsl.h>
float *imsl_f_lin_sol_gen (int n, float a[], float b[],
IMSL_A_COL_DIM, int a_col_dim,
IMSL_TRANSPOSE,
IMSL_RETURN_USER, float x[],
IMSL_FACTOR, int **p_pvt, float **p_factor,
IMSL_FACTOR_USER, int pvt[], float factor[],
IMSL_FAC_COL_DIM, int fac_col_dim,
IMSL_INVERSE, float **p_inva,
IMSL_INVERSE_USER, float inva[],
IMSL_INV_COL_DIM, int inva_col_dim,
IMSL_CONDITION, float *cond,
IMSL_FACTOR_ONLY,
IMSL_SOLVE_ONLY,
IMSL_INVERSE_ONLY,
0)
```

IMSL CNL Manual Organization Example: LIN_SOL_GEN 3/5

Optional Arguments

IMSL_A_COL_DIM, *int a_col_dim (Input)*

The column dimension of the array a.

Default: a_col_dim = *n*

IMSL_TRANSPOSE

Solve *ATx = b.*

Default: Solve *Ax = b*

IMSL_RETURN_USER, *float x[] (Output)*

A user-allocated array of length *n containing the solution x.*

:

Description

The function imsl_f_lin_sol_gen solves a system of linear algebraic equations with a real coefficient matrix A. It first computes the LU factorization of A with partial pivoting such that L-1A = U. The matrix U is upper triangular, while L-1A ≡ Pn Ln-nPn-1… L1P1A ≡ U. The factors Pi and Li are defined by the partial pivoting. Each Pi is an interchange of row i with row j ≥ i. Thus, Pi is defined by that value of j. Every

$$L_i = I + m_i e_i^T$$

is an elementary elimination matrix. The vector mi is zero in entries 1, …, i. This vector is stored as column i in the strictly lower-triangular part of the working array containing the decomposition information.

Visual Numerics

IMSL CNL Manual Organization Example: LIN_SOL_GEN 4/5

Example 1

This example solves a system of three linear equations. This is the simplest use of the function. The equations follow below:

$$x_1 + 3x_2 + 3x_3 = 1$$
$$x_1 + 3x_2 + 4x_3 = 4$$
$$x_1 + 4x_2 + 3x_3 = -1$$

```
#include <imsl.h>
main()
{
int n = 3;
float *x;
float a[] = {1.0, 3.0, 3.0,
1.0, 3.0, 4.0,
1.0, 4.0, 3.0};
float b[] = {1.0, 4.0, -1.0};
/* Solve Ax = b for x */
x = imsl_f_lin_sol_gen (n, a, b, 0);
/* Print x */
imsl_f_write_matrix ("Solution, x, of Ax = b", 1, 3, x, 0);
}
```

IMSL CNL Manual Organization Example: LIN_SOL_GEN 5/5

Output

Solution, x, of Ax = b

1	2	3
-2	-2	3

Warning Errors

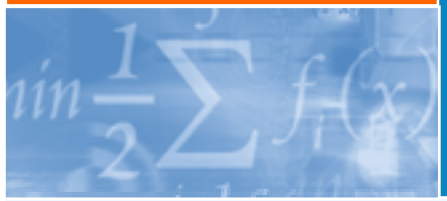
IMSL_ILL_CONDITIONED The input matrix is too ill-conditioned. An estimate of the reciprocal of its *L1 condition number is* "rcond" = #. The solution might not be accurate.

Fatal Errors

IMSL_SINGULAR_MATRIX The input matrix is singular.

How to Get Help?

- Visual Numerics Forums
  - http://www.vni.com.tw/forums/ (Chinese Version)
  - http://forums.vni.com/ (English Version)
- Search the on-line Tips Database
  - http://www.vni.com/tech/imsl/tips.html
- Useful documents in Taiwan Support Center
  - http://www.vni.com.tw/tw/tech/imsl/
- Live Resources
  - Phone Support
    - Open Monday ~ Friday 09:00 AM ~ 18:00 PM
    - Phone Number 886-2-2727-2255
    - Fax Number 886-2-2727-6798
  - Email Support
    - E-mail (Taiwan) imsl@vni.com.tw



How to Compile and Link IMSL CNL

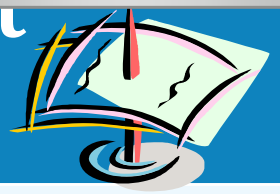
Setup IMSL CNL Environment UNIX

- C shell users:
  - source /opt/vni/imsl/cnl600/itanium/bin/cnlsetup.csh
- Bourne and Korn shell users:
  - . /opt/vni/imsl/cnl600/itanium/bin/cnlsetup.sh
- It is recommended that the setup procedure be executed automatically each time you login rather than having to do it interactively
  - By adding the command (source and .) to the
    - .cshrc (for C shell)
    - .profile (for sh, ksh)
- USTC HP cluster already setup IMSL environment for users

Visual Numerics

Setup IMSL CNL Environment Windows

Windows Only



- Setting MS Visual Studio 2005 environment variables
  - 開始 –> 所有程式 –> Microsoft Visual Studio 2005 –> Visual Studio Tool –> Visual Studio 2005 命令提示字元
- Setting IMSL CNL V6.0 environment variables
  - C:\Program Files\VNI\IMSL\CNL600\vs05PC\BIN\CNLSETUP.bat

Environment Variables

Environment variable	Description
$CC	C compiler
$CFLAGS	C Compiler Options does not include any optimization or debugging options
$LINK_CNL_SHARED	Link options required to link with the shared libraries version of the C Numerical Library
$LINK_CNL_STATIC	Link options required to link with the static libraries version of the C Numerical Library
$LINK_CNL_SHARED_SMP	Link options required to link with the shared libraries version of the C Numerical Library. This option links with the supported vendor BLAS and LAPACK libraries
$LINK_CNL_STATIC_SMP	Link options required to link with the static libraries version of the C Numerical Library. This option links with the supported vendor BLAS and LAPACK libraries
$LINK_CNL	By default, set to $LINK_CNL_SHARED
$LINK_CNL_SMP	By default, set to $LINK_CNL_SHARED_SMP

Visual Numerics

Environment Variables UNIX – IBM Example

- echo $CFLAGS
  - -q64 -DANSI -I/usr/local/imsl/cnl600/rs6000_64/include
  - Does not include any optimization or debugging options
- echo $LINK_CNL
  - -L/usr/local/imsl/cnl600/rs6000_64/lib -brtl -bdynamic -limslcmath_r -limslcmath_scalar_r -limslcmath_iblas_r -limslcstat_r -limslcstat_iblas_r -lm_r
- echo $LINK_CNL_STATIC
  - -L/usr/local/imsl/cnl600/rs6000_64/lib -brtl -bstatic -limslcmath_r -limslcmath_scalar_r -limslcmath_iblas_r -limslcstat_r -limslcstat_iblas_r -limslcmath_r -limslcmath_scalar_r -limslcmath_iblas_r -limslcstat_r -limslcstat_iblas_r -bdynamic -lm_r
- echo $LINK_CNL_SMP
  - -L/usr/local/imsl/cnl600/rs6000_64/lib -brtl -bdynamic -limslcmath_r -limslcmath_smp_r -limslcmath_vblas_r -limslcstat_r -limslcstat_vblas_r **-limsllapack_r** -bstatic **-lesslsmp** -bdynamic -lxlf90_r -lxlsmp -lm_r

Environment Variables Windows

Windows Only

- echo %LINK_CNL_SHARED%
  - -DANSI -MD imslcmath_dll.lib imslcstat_dll.lib msvcrt.lib kernel32.lib user32.lib netapi32.lib advapi32.lib gdi32.lib comdlg32.lib comctl32.lib
- echo %LINK_CNL_STATIC%
  - -DANSI imslcmath.lib imslcmath_scalar.lib imslcmath_iblas.lib imslcstat.lib imslcstat_iblas.lib lmgr.lib libcrvs.lib libsb.lib libcmt.lib oldnames.lib kernel32.lib user32.lib netapi32.lib advapi32.lib gdi32.lib comdlg32.lib comctl32.lib wsock32.lib /link /nodefaultlib:libc.lib /nodefaultlib:libcd.lib /opt:noref
- echo %LINK_CNL_SHARED_SMP%
  - -DANSI -MD imslcmath_mkl_dll.lib imslcstat_mkl_dll.lib msvcrt.lib kernel32.lib user32.lib netapi32.lib advapi32.lib gdi32.lib comdlg32.lib comctl32.lib
- echo %LINK_CNL_STATIC_SMP%
  - -DANSI imslcmath.lib imslcmath_smp.lib imslcmath_vblas.lib imslcstat.lib imslcstat_vblas.lib lmgr.lib libcrvs.lib libsb.lib mkl_c.lib libcmt.lib oldnames.lib kernel32.lib user32.lib netapi32.lib advapi32.lib gdi32.lib comdlg32.lib comctl32.lib wsock32.lib /link /nodefaultlib:libc.lib /nodefaultlib:libcd.lib /opt:noref

Compiling and Linking CNL UNIX

- The following command will compile and link an application program
  - **$CC** -o <executable> **$CFLAGS** <main> **$LINK_CNL**

- $LINK_CNL can easily be replaced by any of the $LINK_CNL* variables
  - $LINK_CNL_SHARED
  - $LINK_CNL_STATIC
  - $LINK_CNL_SHARED_SMP
  - $LINK_CNL_STATIC_SMP

Compiling and Linking CNL Windows – Command Line

Windows Only

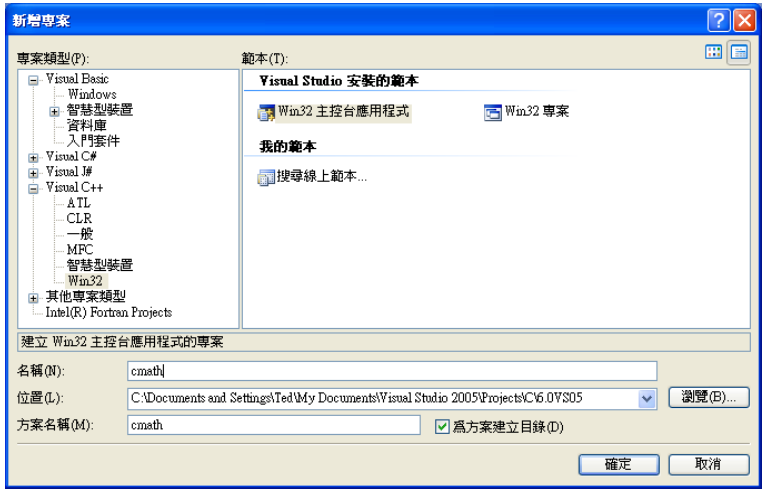
- Link with the DLL libraries version of IMSL CNL
  - CL CMATH.C %LINK_CNL_SHARED%
- Link with DLL libraries version of IMSL CNL and links with the supported vendor BLAS and LAPACK
  - CL CMATH.C %LINK_CNL_SHARED_SMP%
- Link with the static libraries version of IMSL CNL
  - CL CMATH.C %LINK_CNL_STATIC%
- Link with static libraries version of IMSL CNL and links with the supported vendor BLAS and LAPACK
  - CL CMATH.C %LINK_CNL_STATIC_SMP%

Visual Numerics

Compiling and Linking CNL Windows – Visual Studio 2005

Windows Only

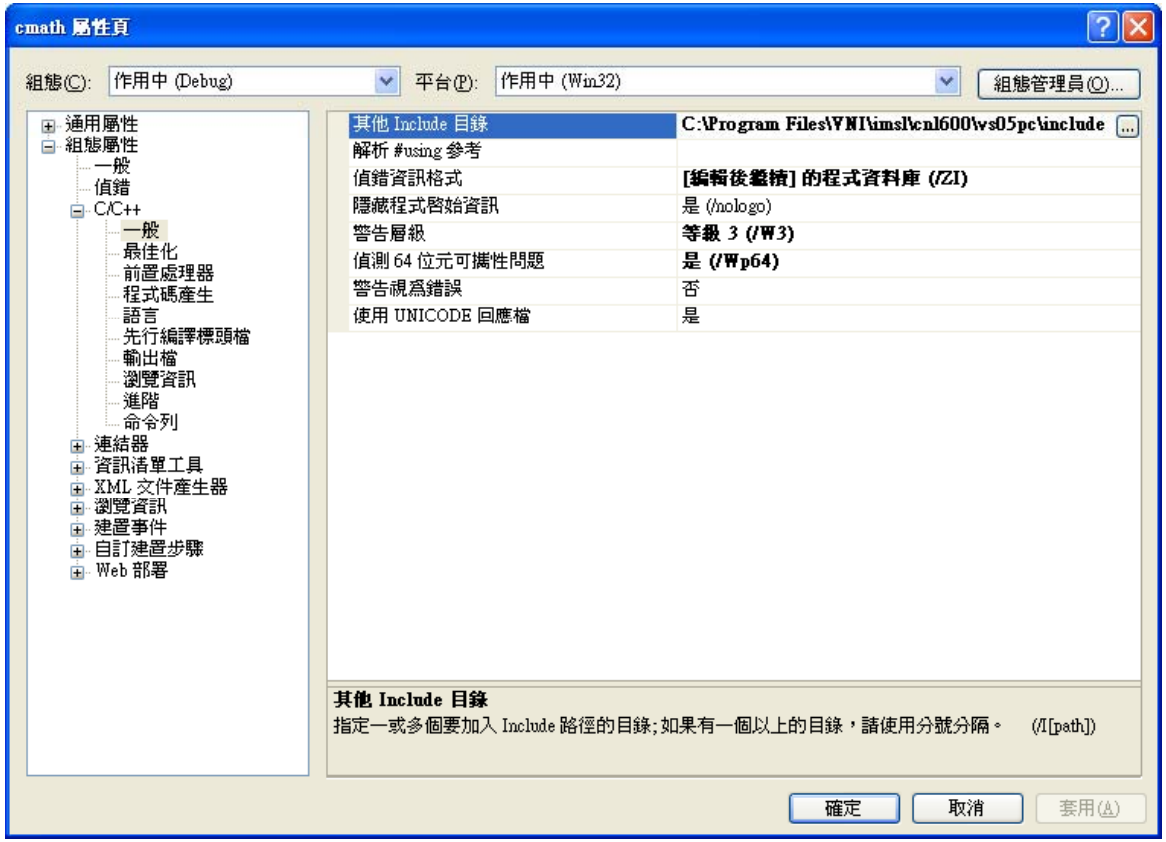
- 檔案 > 新增 > 專案
- Visual C++ > Win32 > Win32 主控台應用程式
- 應用程式設定 > 空專案
- 專案 > 加入現有項目 > C:\Program Files\VNI\imsl\cnl600\vs05pc\examples\validate\cmath.c



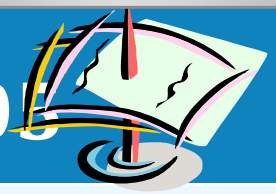
Compiling and Linking CNL Windows – Visual Studio 2005

Windows Only

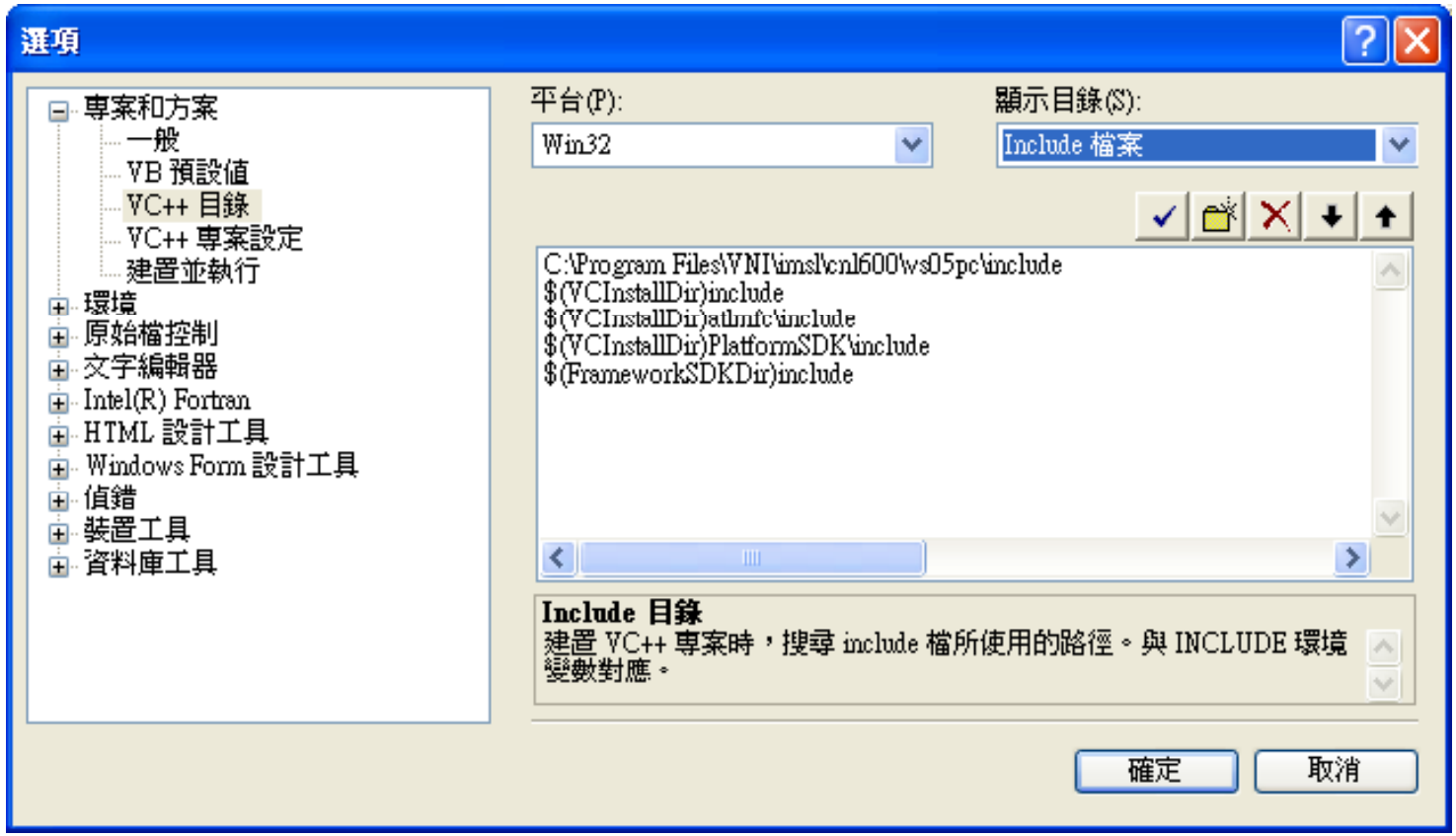
- 專案 > 屬性 > 組態屬性 > C/C++ > 一般 > 其他 Include 目錄 > C:\Program Files\VNI\imsl\cnl600\vs05pc\include



Compiling and Linking CNL Windows – Visual Studio 2005



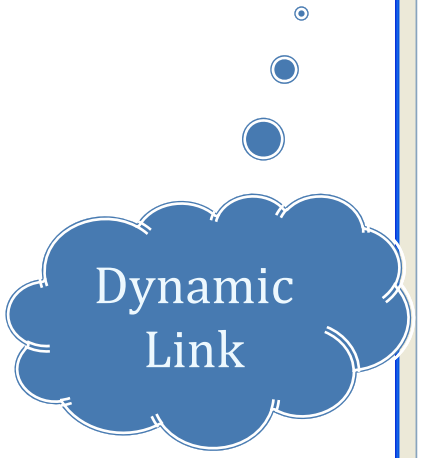
- 工具 > 選項 > 專案和方案 > VC++ 目錄 > Include 檔案

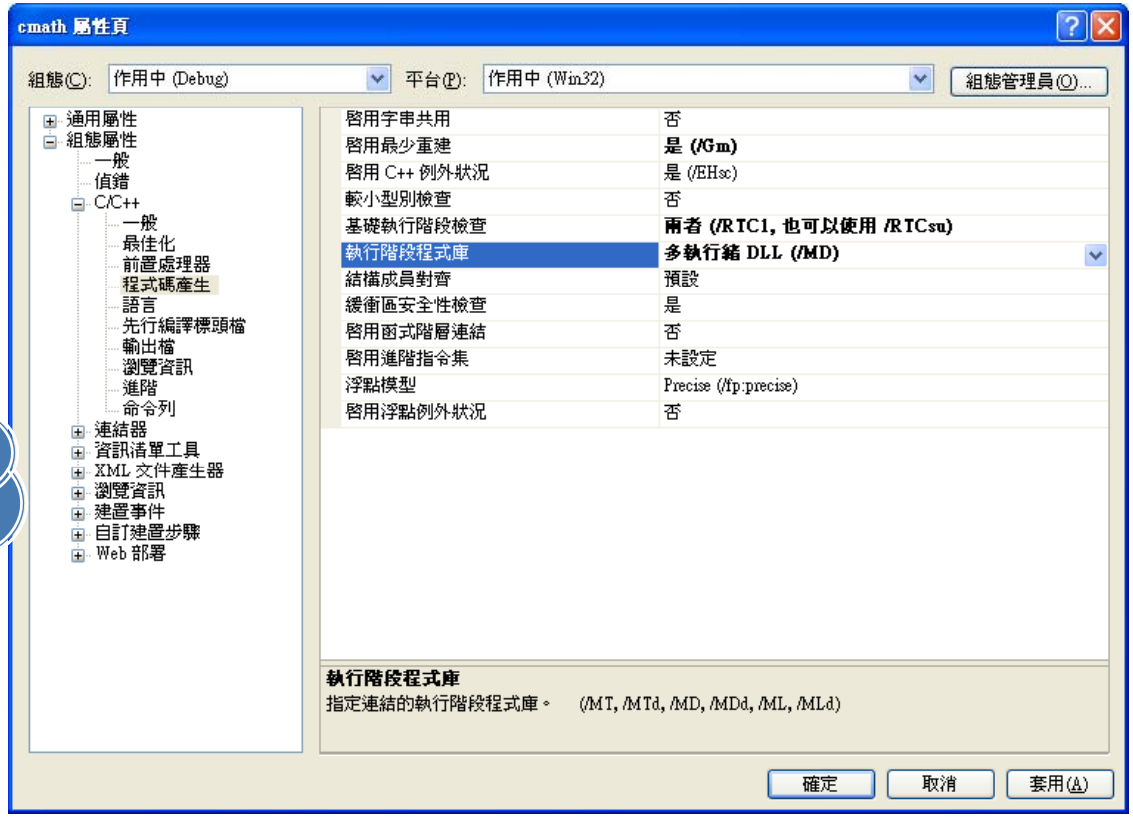


Compiling and Linking CNL Windows – Visual Studio 2005

Windows Only

- 專案 > 屬性 > 組態屬性 > C/C++ > 程式碼產生 > 執行階段程式庫 > 多執行緒 DLL (/MD)

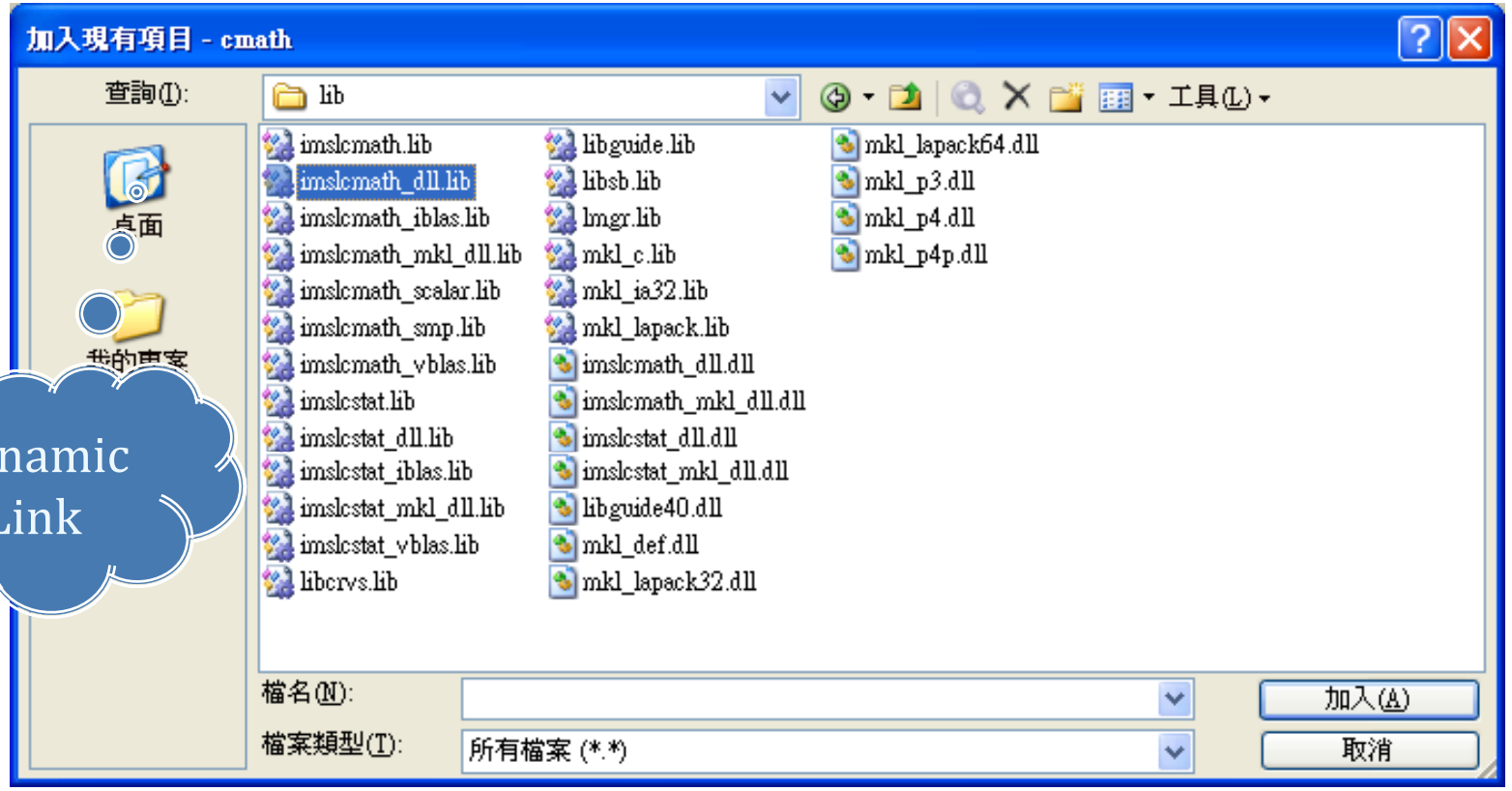




Compiling and Linking CNL Windows – Visual Studio 2005

Windows Only

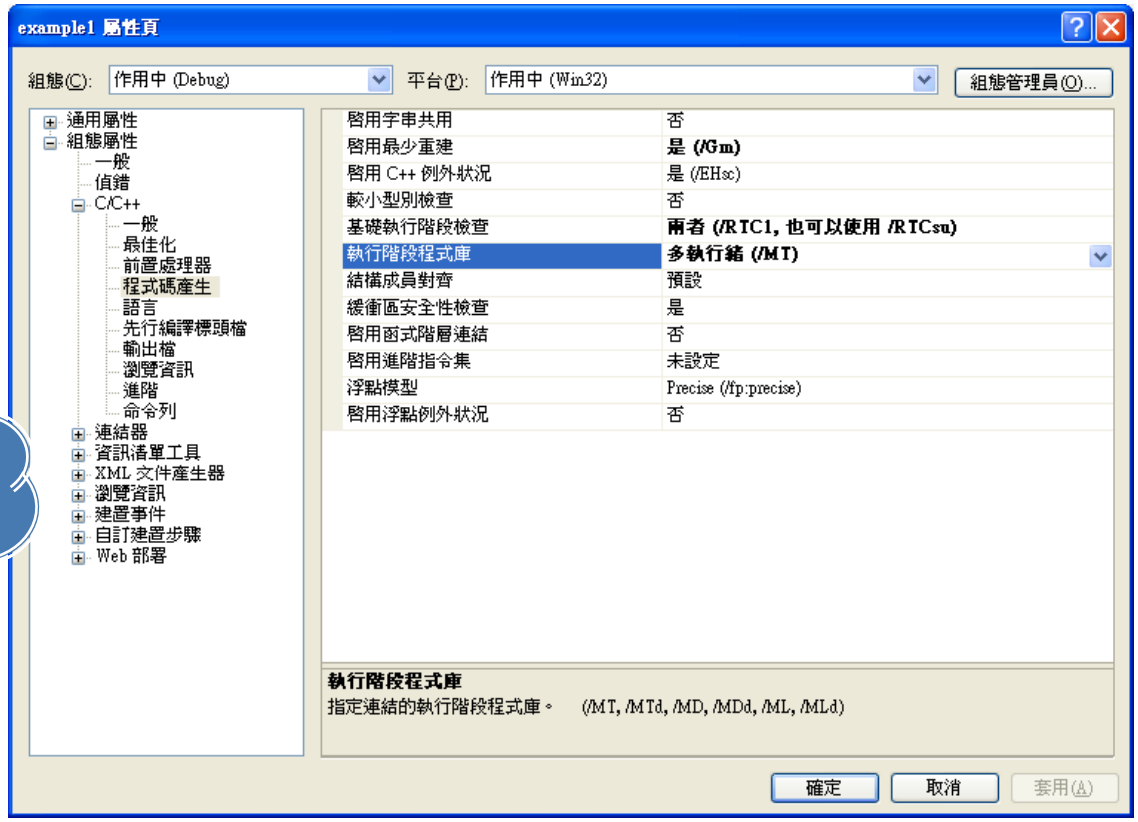
- 專案 > 加入現有項目 > add imslcmath_dll.lib (and/or imslcstat_dll.lib)

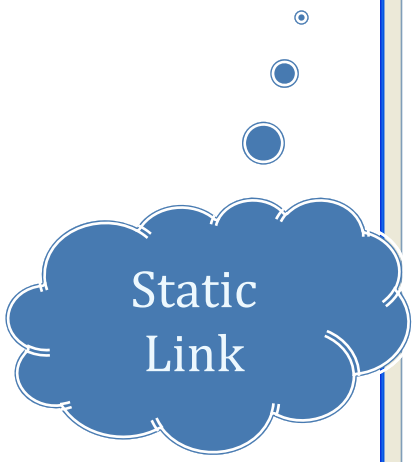


Compiling and Linking CNL Windows – Visual Studio 2005

Windows Only

- 專案 > 屬性 > 組態屬性 > C/C++ > 程式碼產生 > 執行階段程式庫 > 多執行緒 (/MT)



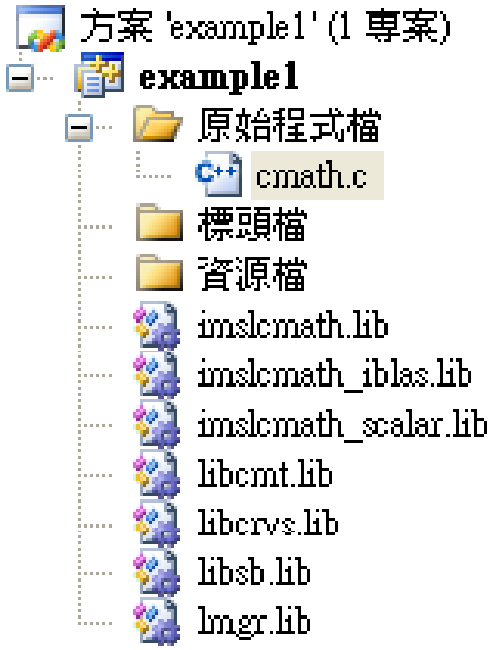


Compiling and Linking CNL Windows – Visual Studio 2005

Windows Only

- 專案 > 加入現有項目 > imslcmath.lib (and/or imslcstat.lib), imslcmath_iblas.lib, imslcmath_scalar.lib, lmgr.lib, libcrvs.lib. Libsb.lib
- 專案 > 加入現有項目 > Program Files\Microsoft Visual Studio 8\VC\lib\libcmt.lib

Static Link

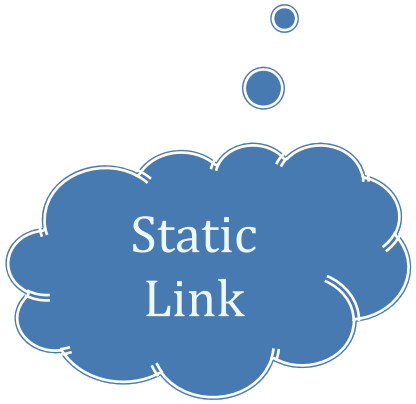


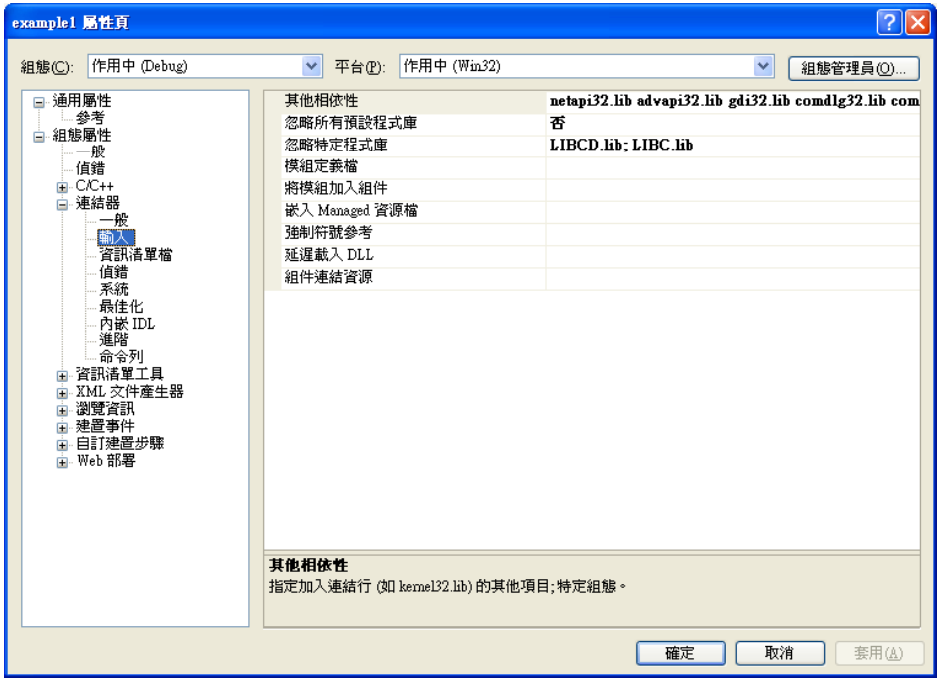
Visual Numerics

Compiling and Linking CNL Windows – Visual Studio 2005

Windows Only

- 專案 > 屬性 > 組態屬性 > 連結器 > 輸入 > 其他相依性 > netapi32.lib advapi32.lib gdi32.lib comdlg32.lib comctl32.lib wsock32.lib
- 專案 > 屬性 > 組態屬性 > 連結器 > 輸入 > 忽略特定程式庫 LIBCD.lib; LIBC.lib

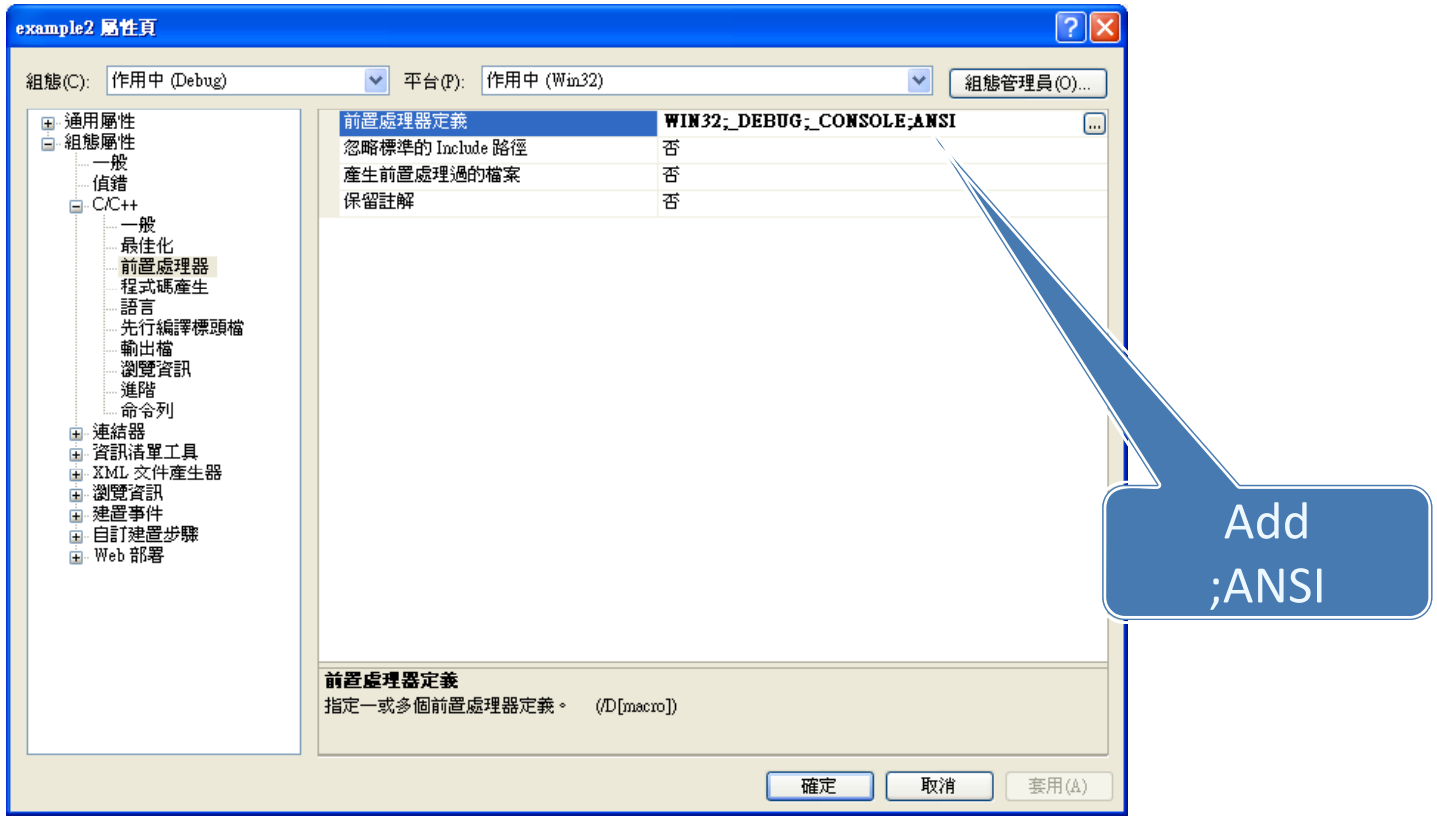




Notice
IMSL CNL on Windows

Windows Only

- 專案 > 屬性 > 組態屬性 > C/C++ > 前置處理器產生 > 前置處理器定義 > ;ANSI



Notice
IMSL CNL on Windows

Windows Only

```
#include <imsls.h>

void main() {
  int *r;
  int seed = 123457;
  float theta = 0.5;
  imsls_random_seed_set (seed);
  r = imsls_random_poisson (5, theta, 0);
  imsls_i_write_matrix ("Poisson(0.5) random deviates", 1, 5, r, 0);
}
```

NO Add ANSI Switch

```
*** TERMINAL Error from imsls_random_poisson.  Illegal optional argument
***          1071644672 on argument number 3.
```

Add ANSI

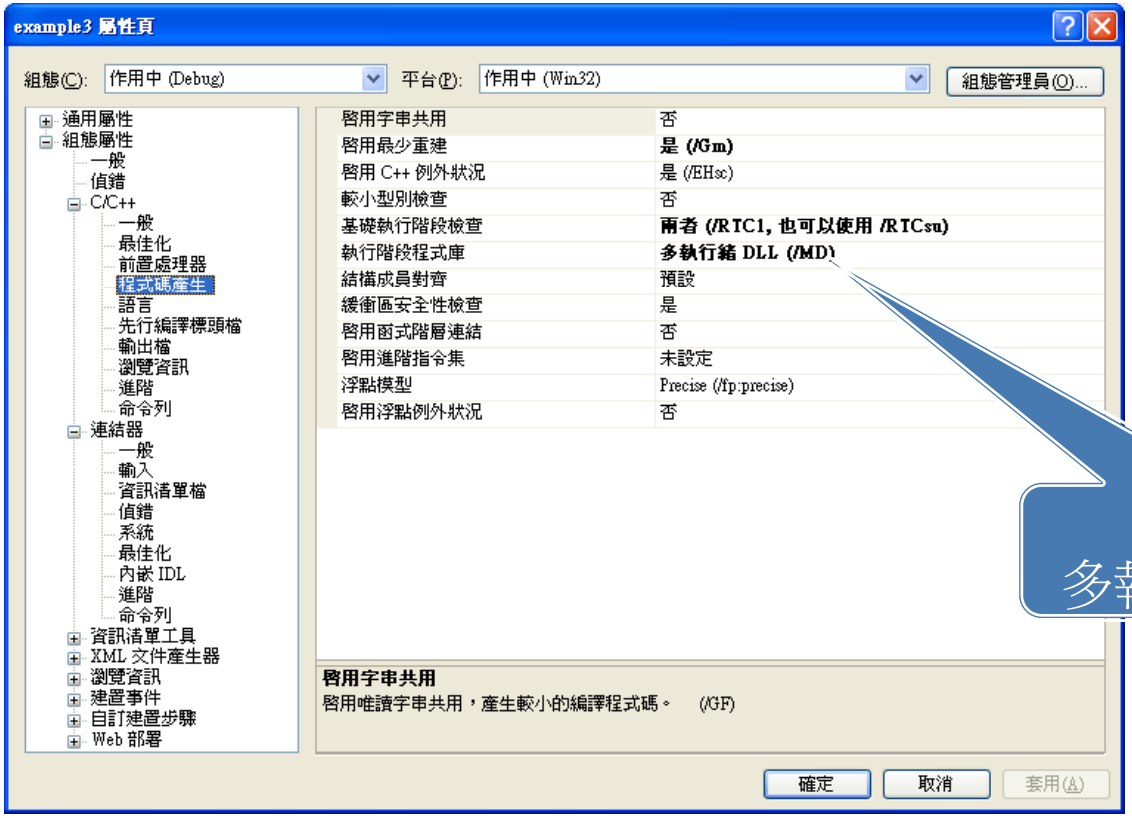
```
Poisson(0.5) random deviates
      1   2   3   4   5
      2   0   1   0   1
```

Visual Numerics

Notice
IMSL CNL on Windows

Windows Only

- 專案 > 屬性 > 組態屬性 > C/C++ > 程式碼產生 > 執行階段程式庫 > 多執行緒 DLL (/MD)



Visual Numerics

Notice
IMSL CNL on Windows

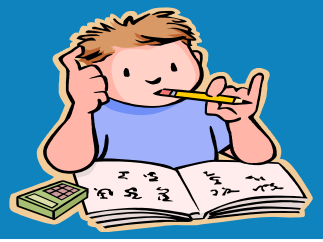
Windows Only

```
#include <stdio.h>
#include <imsl.h>
main() {
  FILE *ofile;
  float x[] = {3.0, 2.0, 1.0};
  imsl_f_write_matrix ("x (default file)", 1, 3, x, 0);
  ofile = fopen("myfile", "w");
  imsl_output_file (IMSL_SET_OUTPUT_FILE, ofile,0);
  imsl_f_write_matrix ("x (myfile)", 1, 3, x, 0);
}
```

```
Using Default
FATAL ERROR

Add /MD
 x (default file)
          1            2            3
          3            2            1
```

Visual Numerics

IMSL C Exercise 1
Validate Program

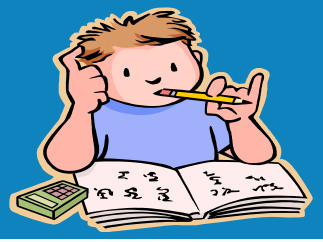


To link and run the validate program

```
cp  /opt/vni/imsl/cnl600/itanium/examples/validate0/validate/cmath.c ~/exercise
$CC  $CFLAGS –o  cmath  cmath.c  $LINK_CNL

./cmath
```

```
 Library version:  IMSL C/Math/Library Version 6.0

       Solution, x of Ax = b
          1            2           3
         -2           -2           3

The next call will generate an error

*** TERMINAL Error from imsl_f_lin_sol_gen.  The order of the matrix must be
***          positive while "n" = -10 is given.
```

IMSL C Exercise 2
Using Optional Arguments



- Solves the transpose problem $A^T x = b$ and return the LU factorization of A with partial pivoting

$$X_1 + 3x_2 + 3x_3 = 1$$
$$X_1 + 3x_2 + 4x_3 = 4$$
$$X_1 + 4x_2 + 3x_3 = -1$$

```
#include <imsl.h>
main()        {
int n = 3, pvt[3];
float factor[9];
float x[3];
float a[] = { 1.0, 3.0, 3.0,
              1.0, 3.0, 4.0,
              1.0, 4.0, 3.0};
float b[] = {1.0, 4.0, -1.0};

imsl_f_lin_sol_gen (n, a, b, IMSL_TRANSPOSE, IMSL_RETURN_USER, x, IMSL_FACTOR_USER, pvt, factor, 0);
imsl_f_write_matrix ("Solution, x, of trans(A)x = b", 1, n, x, 0);
imsl_f_write_matrix ("LU factors of A", n, n, factor, 0);
imsl_i_write_matrix ("Pivot sequence", 1, n, pvt, 0);
}
```

User-allocated containing solution

Solve $A^T x = b$

Pivot sequence for the factorization

LU factorization of A

Visual Numerics

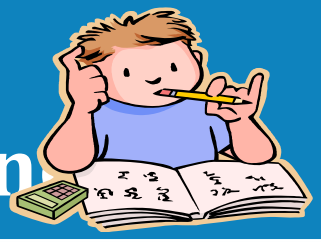
IMSL C Exercise 2
Submit Job

```
#!/bin/sh
#@ job_type=serial
#@ environment = COPY_ALL
#@ initialdir = /home/tedliu/exercise_c
#@ output = poe1.$(jobid).$(stepid).out
#@ error =  poe1.$(jobid).$(stepid).err
#@ wall_clock_limit = 60
#@ class = short
#@ queue
timex ./ex2
sleep 20
```

llsubmit run2.cmd

```
Solution, x, of trans(A)x = b
1       2       3
4      -4       1

LU factors of A
        1       2       3
1       1       3       3
2      -1       1       0
3      -1       0       1

Pivot sequence
1       2       3
1       3       3
```

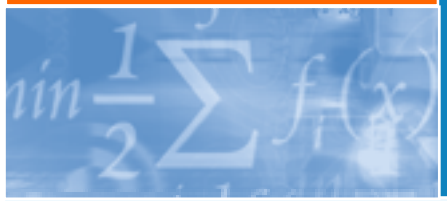
IMSL C Exercise 3
Call Double Precision Routin



- Solves same problem with double data type

$$X_1 + 3x_2 + 3x_3 = 1$$
$$X_1 + 3x_2 + 4x_3 = 4$$
$$X_1 + 4x_2 + 3x_3 = -1$$

```
#include <imsl.h>
main()        {
int n = 3, pvt[3];
double factor[9];
double x[3];
double a[] = {  1.0, 3.0, 3.0,
                1.0, 3.0, 4.0,
                1.0, 4.0, 3.0};
double b[] = {1.0, 4.0, -1.0};

imsl_d_lin_sol_gen (n, a, b, IMSL_TRANSPOSE, IMSL_RETURN_USER, x, IMSL_FACTOR_USER, pvt, factor, 0);
imsl_d_write_matrix ("Solution, x, of trans(A)x = b", 1, n, x, 0);
imsl_d_write_matrix ("LU factors of A", n, n, factor, 0);
imsl_i_write_matrix ("Pivot sequence", 1, n, pvt, 0);
}
```



IMSL CNL Inside

IMSL CNL Error Handling Mechanism

- IMSL CNL functions attempt to detect and report errors and invalid input
- Error are classified according to severity and are assigned a code number

IMSL CNL Error Handling
Error Type and Actions

- IMSL CNL error type and default actions
  - defined in the include file imsl.h and imsls.h

Error Level	Message PRINT	Program Termination	Contents of ERROR
IMSL_NOTE	N	N	Possibility of error
IMSL_ALERT	N	N	Underflow
IMSL_WARNING	Y	N	Program must be modified in case
IMSL_FATAL	Y	Y	Program to be modified
IMSL_TERMINAL	Y	Y	Fatal error
IMSL_WARNING_IMMEDIATE	Y	N	Same as WARNING Message output is prompt
IMSL_FATAL_IMMEDIATE	Y	Y	Same as FATAL Message output is prompt

Visual Numerics

IMSL CNL Error Handling error_options Routines 1/2

- **error_options**
  - Sets various error handling options
- **Synopsis with Optional Arguments**

```
#include <imsl.h>
void imsl_error_options (
IMSL_SET_PRINT, Imsl_error type, int setting,
IMSL_SET_STOP, Imsl_error type, int setting,
IMSL_SET_TRACEBACK, Imsl_error type, int setting,
IMSL_FULL_TRACEBACK, int setting,
IMSL_GET_PRINT, Imsl_error type, int *psetting,
IMSL_GET_STOP, Imsl_error type, int *psetting,
IMSL_GET_TRACEBACK, Imsl_error type, int *psetting,
IMSL_SET_ERROR_FILE, FILE *file,
IMSL_GET_ERROR_FILE, FILE **pfile,
IMSL_ERROR_MSG_PATH, char *path,
IMSL_ERROR_MSG_NAME, char *name
IMSL_ERROR_PRINT_PROC, Imsl_error_print_proc print_proc,
IMSL_SET_SIGNAL_TRAPPING, int setting,
0)
```

IMSL CNL Error Handling error_options Routines 2/2

- **imsl_error_options** (IMSL_SET_PRINT, Imsl_error type, int setting, 0);
  - 0 (zero for all types)
  - IMSL_NOTE
  - IMSL_ALERT
  - IMSL_WARNING
  - IMSL_WARNING_IMMEDIATE
  - IMSL_FATAL
  - IMSL_FATAL_IMMEDIATE
  - IMSL_TERMINAL
  - -1: no change
  - 1: output
  - 0: no output
  - 2: back to default
- Example
  - Continue at FATAL level and print message at ALERT
  - imsl_error_options (**IMSL_SET_STOP**, **IMSL_FATAL**, 0, **IMSL_SET_PRINT**, **IMSL_ALERT**, 1, 0);

IMSL CNL Error Handling error_code Routines

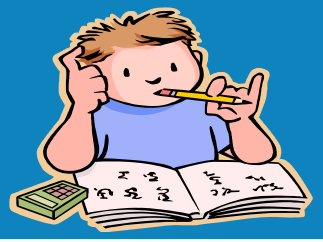
- **error_code**
  - Gets the code corresponding to the error message from the last function called
- **Synopsis**
  - *#include <imsl.h>*
  - *long imsl_error_code ( )*
- **Return Value**
  - This function returns the error message code from the last IMSL function called

- The include file imsl.h and imsls.h defines a name for each error code
- Example

```
x = imsl_f_lin_sol_gen (n, a, b, 0);
If  (imsl_error_code() == IMSL_SINGULAR_MATRIX)  {
    process at error
}
```

lin_sol_gen defines
FATAL Error: The input matrix is singular

IMSL C Exercise 4
IMSL Error Handling



```
#include <stdio.h>
#include "imsl.h“
int main (int argc, char* argv[])   {
        float res;
        long code;

        imsl_error_options (IMSL_SET_STOP, IMSL_TERMINAL, 0, 0);
        res = imsl_f_beta (-1.0, .5);
        code = imsl_error_code ();
        printf ("Error Code = %d\n", code);
        printf ("Run to here....\n\n");
}
```

Evaluates the complete beta function
float imsl_f_beta (float a, float b)

```
*** TERMINAL Error from imsl_f_beta.  Both "x" = -1.000000e+00 and "y" =
***          5.000000e-01 must be greater than zero.
```

Without error_options code

```
*** TERMINAL Error from imsl_f_beta.  Both "x" = -1.000000e+00 and "y" =
***          5.000000e-01 must be greater than zero.

Error Code = 9031
Run to here....
```

With error_options code

IMSL CNL Error Handling
Customize Error Message

- Error messages file and utilities located at
  - /usr/local/imsl/cnl600/rs6000_64/bin
  - Error messages raw data
    - imslerr.dat / imsls_e.dat
  - Error utility
    - imslerr and imsls_e
  - Error message file
    - imslerr.bin / imsls_e.bin
  - Environment variable
    - IMSLERRPATH
    - IMSLSERRPATH

```
:
9030  IMSL_ERF_ALGORITHM     Sufficiently accurate result was not
                             &  achieved. An algorithmic problem
                             & exists.
9031  IMSL_BETA_NEG_ARG      Both "x" = %(F1) and "y" = %(F2) must be
                             & greater than zero. :
:
```

IMSL Underflow and Overflow

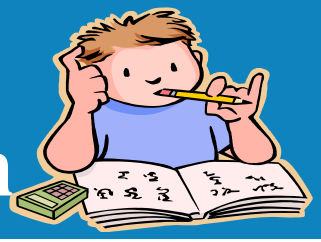
- Underflow
  - IMSL codes are written so that computations are not affected by underflow
  - System places a zero in the register
  - Error level is **IMSL_ALERT**

- Overflow
  - IMSL codes are written to avoid overflow
  - A program that produces system error message indicating overflow should be examined for programming errors
  - Error level is **IMSL_FATAL**
  - Example of error code
    - IMSL_LARGE_ARG_OVERFLOW
    - IMSL_ZERO_ARG_OVERFLOW
    - IMSL_SMALL_ARG_OVERFLOW

Missing Values

- Some functions in the IMSL CNL allow the data to contain missing values
- These functions recognize as a missing value the special value referred to as “Not a Number” or NaN
- The actual missing value is different on different computers
  - It can be obtained by reference to the function machine

IMSL C Exercise 5
Get Computer's Information

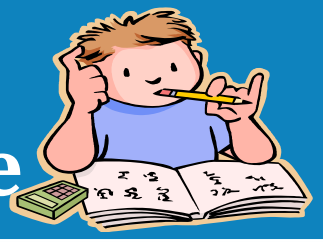


```
#include <imsl.h>
main() {
  int n;
  float fans;
  double dans;
  for (n = 1; n <= 8; n++) {
    fans = imsl_f_machine (n);
    printf("imsl_f_machine(%d) = %g\n", n, fans);
  }
  for (n = 1; n <= 8; n++) {
    dans = imsl_d_machine (n);
    printf("imsl_d_machine(%d) = %g\n", n, dans);
  }
}
```

n	Definition
1	$B^{E_{min_f}-1}$, the smallest positive number
2	$B^{E_{max_f}}(1-B^{-N_f})$, the largest number
3	B^{-N_f}, the smallest relative spacing
4	B^{1-N_f}, the largest relative spacing
5	$\log_{10}(B)$
6	NaN (not a number)
7	positive machine infinity
8	negative machine infinity

```
imsl_f_machine(1) = 1.17549e-38
imsl_f_machine(2) = 3.40282e+38
imsl_f_machine(3) = 5.96046e-08
imsl_f_machine(4) = 1.19209e-07
imsl_f_machine(5) = 0.30103
imsl_f_machine(6) = NaNQ
imsl_f_machine(7) = INF
imsl_f_machine(8) = -INF

imsl_d_machine(1) = 2.22507e-308
imsl_d_machine(2) = 1.79769e+308
imsl_d_machine(3) = 1.11022e-16
imsl_d_machine(4) = 2.22045e-16
imsl_d_machine(5) = 0.30103
imsl_d_machine(6) = NaNQ
imsl_d_machine(7) = INF
imsl_d_machine(8) = -INF
```

IMSL C Exercise 6
Data Contains Missing Value



```
#include <imsls.h>
#define N_OBSERVATIONS 10
#define N_VARIABLES 3
main() {
int n_keys=2;
float x[N_OBSERVATIONS][N_VARIABLES] = {
1.0, 1.0, 1.0,
2.0, 1.0, 2.0,
1.0, 1.0, 3.0,
1.0, 1.0, 4.0,
2.0, 2.0, 5.0,
1.0, 2.0, 6.0,
1.0, 2.0, 7.0,
1.0, 1.0, 8.0,
2.0, 2.0, 9.0,
1.0, 1.0, 9.0};
x[4][1]=imsls_f_machine(6);
x[6][0]=imsls_f_machine(6);
imsls_f_sort_data (N_OBSERVATIONS, N_VARIABLES, x, n_keys, 0);
imsls_f_write_matrix("sorted x", N_OBSERVATIONS, N_VARIABLES, (float *)x, 0);
}
```

sort_data

Sorts observations by specified keys, with option to tally cases into a multi-way frequency table.

Synopsis

#include <imsls.h>

void imsls_f_sort_data (int n_observations, int n_variables, float x[], int n_keys, ..., 0)

- **Int n_observations** (Input) Number of observations (rows) in x.
- **int n_variables** (Input) Number of variables (columns) in x.
- **float x[]** (Input/Output) An n_observations × n_variables matrix containing the observations to be sorted. The sorted matrix is returned in x
- **int n_keys** (Input) Number of columns of x on which to sort. The first n_keys columns of x are used as the sorting eys

```
                sorted x
              1         2         3
1             1         1         1
2             1         1         9
3             1         1         3
4             1         1         4
5             1         1         8
6             1         2         6
7             2         1         2
8             2         2         9
9      .........        2         7
10            2  .........        5
```

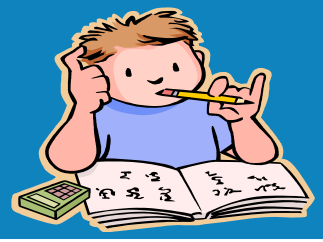
IMSL CNL Thread Safe Usage Overview

- IMSL CNL can be safely called from a multithreaded application
  - Support either **POSIX** threads or **WIN32** threads
- IMSL CNL is used in a multithreaded application, the calling program must adhere
  1. Signal handling
  2. Routines that Produce Output
  3. Input Arguments

IMSL CNL Thread Safe Usage Signal Handling

- When calling IMSL CNL from a multithreaded application it is necessary to turn IMSL signal-handling capability off
- The IMSL signal-trapping mechanism must be disabled before any threads are created
  - **imsl_error_options (IMSL_SET_SIGNAL_TRAPPING, 0, 0)**;

IMSL C Exercise 7
Turn Off Signal-Trapping



IMSL signal-trapping mechanism must be disabled before any threads are created

```
#include <pthread.h>
#include <stdio.h>
#include "imsl.h"
void *ex1(void* arg);
void *ex2(void* arg);
void main() {
  pthread_t thread1;
  pthread_t thread2;

  imsl_error_options (IMSL_SET_SIGNAL_TRAPPING, 0, 0);

  pthread_create(&thread1, NULL ,ex1, (void *)NULL);
  pthread_create(&thread2, NULL ,ex2, (void *)NULL);
  pthread_join(thread1, NULL);
  pthread_join(thread2, NULL);
}
```

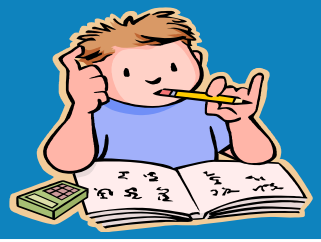
```
void *ex1(void *arg) {
  float res;
  res = imsl_f_beta (0.5, 0.2);
  printf("Thread 1 %f \n", res);
}
void *ex2(void *arg) {
  float res;
  res = imsl_f_gamma (1.5);
  printf("Thread 2 %f \n", res);
}
```

```
Thread 2 0.886227
Thread 1 6.268653
```

Without SIGNAL TRAPPING

```
*** FATAL    Error IMSL_NO_MT_SIGNAL_HANDLING from imsl_f_beta.
***          C/Math/Library signal handling must be turned off when using
***          multiple threads.  See imsl_err_options for instructions on
***          turning off signal handling by C/Math/Library.
:
```

IMSL C Exercise 8
Set Thread's Error Handler



Each thread can set its own options for IMSL CNL error handler

```
#include <pthread.h>
#include <stdio.h>
#include "imsl.h"
void *ex1(void* arg);
void *ex2(void* arg);
void main() {
  pthread_t thread1;
  pthread_t thread2;

  imsl_error_options (IMSL_SET_SIGNAL_TRAPPING, 0, 0);

  if (pthread_create(&thread1, NULL ,ex1, (void *)NULL) != 0)
  perror("pthread_create"), exit(1);
  if (pthread_create(&thread2, NULL ,ex2, (void *)NULL) != 0)
  perror("pthread_create"), exit(1);

  if (pthread_join(thread1, NULL) != 0)
  perror("pthread_join"),exit(1);
  if (pthread_join(thread2, NULL) != 0)
  perror("pthread_join"),exit(1);
}
```

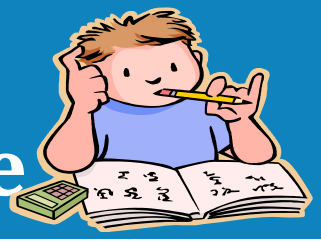
```
void *ex1(void* arg) {
  float res;
  imsl_error_options (IMSL_SET_STOP, IMSL_TERMINAL, 0, 0);
  res = imsl_f_beta(-1.0, .5);
}

void *ex2(void* arg) {
  float res;
  imsl_error_options (IMSL_SET_STOP, IMSL_TERMINAL, 0,
        IMSL_SET_TRACEBACK, IMSL_TERMINAL, 1, 0);
  res = imsl_f_gamma(-1.0);
}
```

```
*** TERMINAL Error from imsl_f_beta.  Both "x" = -1.000000e+00 and "y" =
***       5.000000e-01 must be greater than zero.


*** TERMINAL Error from imsl_f_gamma.  The argument for the function can not
***       be a negative integer. Argument "x" = -1.000000e+00.

Here is a traceback of the calls in reverse order.
 Error Type                Error Code                Routine
 ----------                ----------                -------
 IMSL_TERMINAL             IMSL_NEGATIVE_INTEGER     imsl_f_gamma
                                                     USER
```

IMSL CNL Thread Safe Usage Produce Output

- The function **imsl_output_file** can be used to control which file the output is directed
- Each thread must call **imsl_output_file** to change the default setting of where output will be directed

IMSL C Exercise 9
Change Thread's Output File



Each thread must call imsl_output_file to change the default setting of where output will be directed

```
#include <pthread.h>
#include <stdio.h>
#include "imsl.h"
void *ex1(void* arg);
void *ex2(void* arg);
void main() {
  pthread_t thread1;
  pthread_t thread2;
  imsl_error_options(IMSL_SET_SIGNAL_TRAPPING, 0, 0);
  pthread_create(&thread1, NULL ,ex1, (void *)NULL);
  pthread_create(&thread2, NULL ,ex2, (void *)NULL);
  pthread_join(thread1, NULL);
  pthread_join(thread2, NULL);
}

void *ex1(void* arg) {
  float *rand_nums = NULL;
  FILE *file_ptr;
  file_ptr = fopen("thread1.out", "w");
  imsl_output_file (IMSL_SET_OUTPUT_FILE, file_ptr, 0);
  imsl_random_seed_set(12345);
  rand_nums = imsl_f_random_uniform (5, 0);
  imsl_f_write_matrix ("Random Numbers", 5, 1, rand_nums, 0);
  fclose(file_ptr);
}
void *ex2(void* arg) {
  int n = 3;
  float *x;
  float a[] = {1.0, 3.0, 3.0,
               1.0, 3.0, 4.0,
               1.0, 4.0, 3.0};
  float b[] = {1.0, 4.0, -1.0};
  FILE *file_ptr;

  file_ptr = fopen ("thread2.out", "w");
  imsl_output_file (IMSL_SET_OUTPUT_FILE, file_ptr, 0);
  x = imsl_f_lin_sol_gen (n, a, b, 0);
  imsl_f_write_matrix ("Solution, x, of Ax = b", 1, 3, x, 0);
  fclose(file_ptr);
}
```

```
>cat thread1.out
Random Numbers
 1      0.0966
 2      0.8340
 3      0.9477
 4      0.0359
 5      0.0115

>cat thread2.out

      Solution, x, of Ax = b
         1           2           3
        -2          -2           3
```

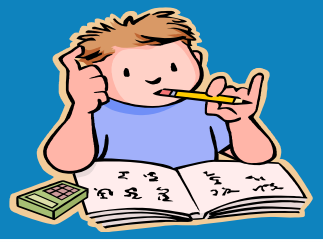
IMSL CNL Thread Safe Usage Input Arguments

- Some arguments that may appear to be input-only are temporarily modified during the call and restored before returning to the caller
- Avoid usage of the same data space in separate threads calling functions in IMSL CNL

Memory Allocation for Output 1/2

- Many functions return a pointer to an array
- **IMSL_RETURN_USER**, float a[]
  - The answers are stored in the user-provided array, and the pointer returned by the function is set to point to the user-provided array
  - If an invocation does not use IMSL_RETURN_USER, then the function initializes the pointer (through a memory allocation request to *malloc*) and stores the answers there

IMSL C Exercise 10
IMSL Routine's Answer



```
#include <imsl.h>
main()        {
  int n = 3;
  float x1[3];
  float *x2;
  float a[] = { 1.0, 3.0, 3.0,
                1.0, 3.0, 4.0,
                1.0, 4.0, 3.0};
  float b[] = {1.0, 4.0, -1.0};

  imsl_f_lin_sol_gen (n, a, b, IMSL_RETURN_USER, x1, 0);
  x2 = imsl_f_lin_sol_gen (n, a, b, 0);

  imsl_f_write_matrix ("Solution 1", 1, n, x1, 0);
  imsl_f_write_matrix ("Solution 2", 1, n, x2, 0);
}
```

```
                Solution 1
          1             2             3
         -2            -2             3

                Solution 2
          1             2             3
         -2            -2             3
```

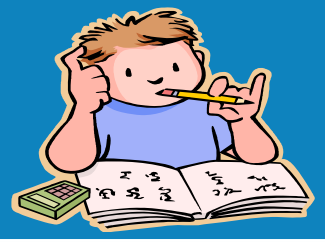
User provided array

Initializes a pointer

Memory Allocation for Output 2/2

- Some optional arguments specify whether additional computed output arrays
  - Allocated by the user
  - Allocated internally by the function
- Example
  - lin_sol_gen
    - IMSL_INVERSE_USER, float inva[]
    - IMSL_INVERSE, float **p_inva
  - Allocated by the user
    - float inva[9];
    - imsl_f_lin_sol_gen (n, a, b, IMSL_INVERSE_USER, inva,…);
  - Allocated internally by the function
    - float *p_inva;
    - imsl_f_lin_sol_gen (n, a, b, IMSL_INVERSE, &p_inva, …);

IMSL C Exercise 11
Computation Output Array



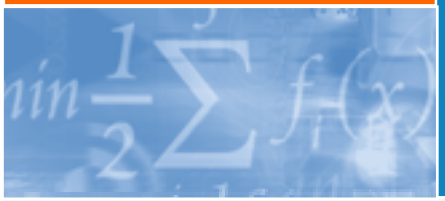
```
#include <imsl.h>
main()  {
  int n = 3;
  float *x;
  float a[] = { 1.0, 3.0, 3.0,
                1.0, 3.0, 4.0,
                1.0, 4.0, 3.0};
  float b[] = {1.0, 4.0, -1.0};

  float *inva1;
  float inva2[9];

  x = imsl_f_lin_sol_gen (n, a, b, IMSL_INVERSE, &inva1, 0);
  x = imsl_f_lin_sol_gen (n, a, b, IMSL_INVERSE_USER, inva2, 0);

  imsl_f_write_matrix ("Solution 1", n, n, inva1, 0);
  imsl_f_write_matrix ("Solution 2", n, n, inva2, 0);
}
```

IMSL_INVERSE
inverse of the matrix *A*

Initializes a pointer

User provided array

```
                Solution 1
                1           2           3
1               7          -3          -3
2              -1           0           1
3              -1           1           0

                Solution 2
                1           2           3
1               7          -3          -3
2              -1           0           1
3              -1           1           0
```



Printing Results

- Most IMSL CNL functions do not print any of the results, the output is returned in C variables
- **write_matrix**
  - Prints a rectangular matrix (or vector) stored in contiguous memory locations
- **Synopsis**

  #include <imsl.h>

  void **imsl_f_write_matrix** (char *title, int nra, int nca, float a[], …, 0)

  For int a[], use **imsl_i_write_matrix**

  For double a[], use **imsl_d_write_matrix**

  For f_complex a[], use **imsl_c_write_matrix**

  For d_complex a[], use **imsl_z_write_matrix**
- **Required Arguments**

  char *title (Input)

  The matrix title. Use \n within a title to create a new line. Long titles are automatically wrapped

  int nra (Input)

  The number of rows in the matrix

  int nca (Input)

  The number of columns in the matrix

  float a[] (Input)

  Array of size nra × nca containing the matrix to be printed

IMSL CNL
Complex Data Types

- Users can perform computations with complex arithmetic by using IMSL predefined data types
  - f_complex
  - d_complex
- The structure is declared as

```
typedef struct {
    float re;
    float im;
} f_complex;
```

IMSL CNL
Complex Functions

Operation	Function Name	Result
z = x + y	z = imsl_c_add(x,y)	f_complex
z = x (drop precision)	z = imsl_cz_convert(x)	f_complex
z = a + ib	z = imsl_cf_convert(a,b)	f_complex
a = |z|	a = imsl_c_abs(z)	float
z = cos(x)	z = imsl_c_cos(z)	f_complex
$z = x^a$	z = imsl_cf_power(x,a)	f_complex
:	:	:

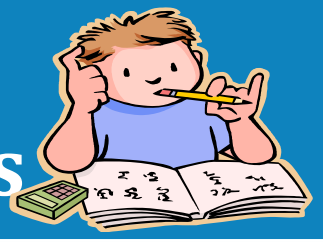
Parallelism in the IMSL CNL

- IMSL CNL employs fine-grain parallelism to take advantage of multiple CPUs
- Fine-grain parallelism
  - Within a subroutine or function and is limited to shared-memory (SMP)
  - Done at the DO loop level
- Parallelism is seamless to the user
  - Only the environment variables need to be set and used

IMSL CNL 6.0 Performance Enhancements 1/2

- Performance increases can be realized by taking advantage of vendor supplied BLAS, FFT and LAPACK routines (**IBM ESSL 4.2**)
  - LINK_CNL_SHARED_SMP
    - link with the shared library
  - LINK_CNL_STATIC_SMP
    - link with the static library
- IMSL CNL also uses IBM ESSL library to take advantage of multiple CPUs
  - Set **OMP_NUM_THREADS**=<number of threads to use>
  - Default is 1

IMSL C Exercise 12
Performance Enhancements



```
#include <imsl.h>
#include <stdio.h>
#define N 3000
void main() {
  float *x;
  float *a, *b;
  clock_t start, finish;

  a = imsl_f_random_uniform (N*N, 0);
  b = imsl_f_random_uniform (N, 0);

  start = clock();
  x = imsl_f_lin_sol_gen (N, a, b, 0);
  finish = clock();
  printf("Used %.2f seconds.\n",
         (double)(finish - start)/CLOCKS_PER_SEC);
}
```

N =3000

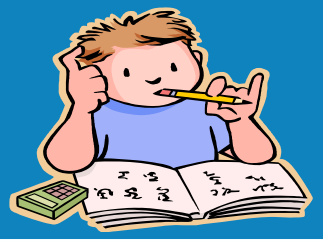
N =4000

```
$CC -o ex12 ex12.c $CFLAGS $LINK_CNL
./ex12
Used 19.88 seconds.

$CC -o ex12 ex12.c $CFLAGS $LINK_CNL_SMP
./ex12
Used 5.45 seconds.
```

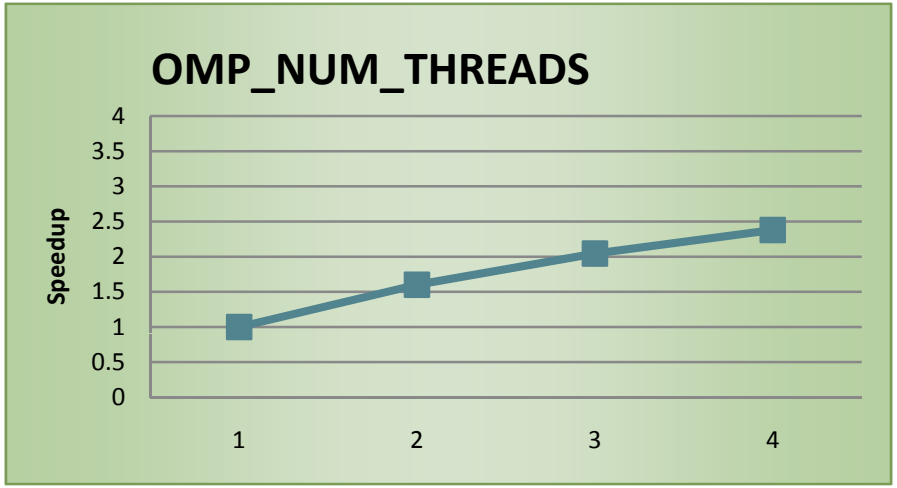
```
$CC -o ex12 ex12.c $CFLAGS $LINK_CNL
./ex12
Used 49.92 seconds.

$CC -o ex12 ex12.c $CFLAGS $LINK_CNL_SMP
./ex12
Used 12.71 seconds.
```

Visual Numerics

IMSL C Exercise 13
IMSL CNL Support SMP

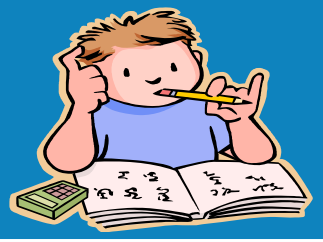


```
#include <imsl.h>
#include <stdio.h>
#define N 5000
void main() {
  float *x;
  float *a, *b;
  a = imsl_f_random_uniform (N*N, 0);
  b = imsl_f_random_uniform (N, 0);

  x = imsl_f_lin_sol_gen (N, a, b, 0);
}
```




```
export OMP_NUM_THREADS=1
timex ./ex13
real 25.80
user 25.72
sys  0.00

export OMP_NUM_THREADS=2
timex ./ex13
real 16.13
user 26.60
sys  0.10

export OMP_NUM_THREADS=3
timex ./ex13
real 12.62
user 26.56
sys  0.15

export OMP_NUM_THREADS=4
timex ./ex13
real 10.84
user 26.59
sys  0.25
```

IMSL C Exercise 14
Submit Job for SMP



```
#include <imsl.h>
#include <stdio.h>
#define N 5000
void main() {
  float *x;
  float *a, *b;

  a = imsl_f_random_uniform(N*N, 0);
  b = imsl_f_random_uniform(N, 0);

  x = imsl_f_lin_sol_gen (N, a, b, 0);
}
```

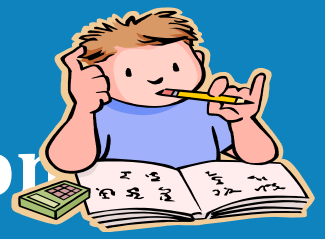
```
#!/bin/sh
#@ job_type=serial
#@ environment = COPY_ALL
#@ initialdir = /home/tedliu/exercise_c
#@ output = poe1.$(jobid).$(stepid).out
#@ error =  poe1.$(jobid).$(stepid).err
#@ wall_clock_limit = 60
#@ class = short
#@ environment = OMP_NUM_THREADS=4
#@ queue
timex ./ex14
sleep 20
```

```
llsubmit run14.cmd
llsubmit: The job "i82.2737" has been submitted.
cat poe1.2737.0.err

real 9.37
user 22.80
sys  0.17
```

User-Supplied Function

- Some IMSL CNL routines accept user-supplied function
- Example **min_uncon**
  - Find the minimum point of a smooth function f(x) of a single variable using only function evaluations
  - Synopsis
    - float imsl_f_min_uncon (float fcn(), float a, float b, …, 0)
  - Required Arguments
    - float imsl_f_min_uncon (float fcn(), float a, float b,.. , 0)
    - float fcn(float x)
      - User-supplied function to compute
    - float a
      - The lower endpoint of the interval
    - float b
      - The upper endpoint of the interval

IMSL C Exercise 15
Solve User Supplied Equation



Optimization Function : Minimum point of f(x) = ex – 5x

```
#include <imsl.h>
#include <math.h>
float fcn(float);
void main () {
  float a = -100.0;
  float b = 100.0;
  float fx, x;
  x = imsl_f_min_uncon (fcn, a, b, 0);
  fx = fcn(x);
  printf ("The solution is: %8.4f\n", x);
  printf ("The function evaluated at the solution is: %8.4f\n", fx);
}

float fcn(float x) {
  return exp(x) - 5.0*x;
}
```

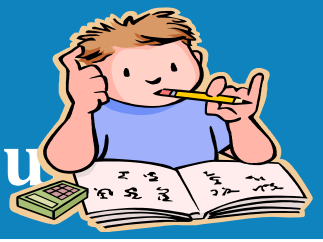
lower (-100) and upper (100) endpoint of the interval

```
The solution is: 1.6094
The function evaluated at the solution is: -3.0472
```

Passing Data to User-Supplied Function

- User-supplied functions requires data that is local to the user's calling function
- User wants to avoid using global data to allow the user-supplied function to access the data
  - Optional arguments can accept an alternative user-supplied function, along with a pointer to the data
  - **IMSL_FCN_W_DATA, float fcn(float x, void *data)**

IMSL C Exercise 16
Passing Data to User-Supplied Fu



Optimization Function : Minimum point of f(x) = ex – 5x

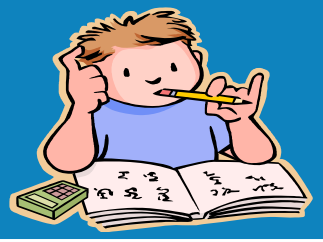
```
#include "imsl.h"
#include <math.h>
static float fcn_w_data(float x, void *data_ptr);
static float fcn(float);
void main() {
  float a = -100.0;
  float b = 100.0;
  float fx, x;
  float usr_data[] = {5.0};
  x = imsl_f_min_uncon (fcn, a, b, IMSL_FCN_W_DATA, fcn_w_data, usr_data, 0);
  fx = fcn_w_data (x, (void*)usr_data);
  printf ("The solution is: %8.4f\n", x);
  printf ("The function evaluated at the solution
}
```

```
static float fcn_w_data(float x, void *data_ptr)  {
  return exp(x) - (*((float*)data_ptr))* x;
}
static float fcn(float x) {
  return;
}
```

IMSL Quadrature
Example - Multivariate Quadrature

- **int_fcn_2d**
  - Computes a two-dimensional iterated integral.
- **Synopsis**
  - float imsl_f_int_fcn_2d (float fcn(), float a, float b, float gcn (float x), float hcn (float x), …, 0)
- **Required Arguments**
  - float fcn (float x, float y) (Input)
    - User-supplied function to be integrated
  - float a (Input)
    - Lower limit of outer integral
  - float b (Input)
    - Upper limit of outer integral
  - float gcn (float x) (Input)
    - User-supplied function to evaluate the lower limit of the inner integral
  - float hcn (float x) (Input)
    - User-supplied function to evaluate the upper limit of the inner integral
- **Return Value**
  - The value of $\int_a^b \int_{gcn(x)}^{hcn(x)} fcn(x,y)\,dy\,dx$

IMSL C Exercise 17
Quadrature Example



Compute the value of the integral

```
#include <math.h>
#include <imsl.h>
float fcn(float x, float y), gcn(float x), hcn(float x);
main() {
  float q, exact;
  q = imsl_f_int_fcn_2d (fcn, 0.0, 1.0, gcn, hcn, 0);
  exact = 0.5*(cos(9.0)+cos(2.0)-cos(10.0)-cos(1.0));
  printf("integral = %10.3f\nexact = %10.3f\n", q, exact);
}
float fcn (float x, float y) {
  return y * cos(x+y*y);
}
float gcn (float x) {
  return 1.0;
}
float hcn (float x) {
  return 3.0;
}
```

Lower (0) and upper (1) limit of outer integral

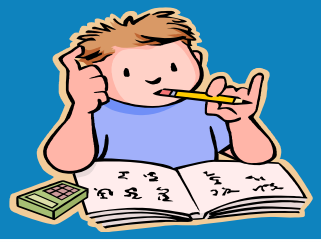
$$\int_0^1 \int_1^3 y\cos(x+y^2)\,dy\,dx$$

```
integral =  -0.514
exact =  -0.514
```

IMSL Differential Equations
Example – ODE solution of initial-value

- **ode_runge_kutta**
  - Solves an initial-value problem for ordinary differential equations using the Runge-Kutta-Verner
- **Synopsis**
  - float imsl_f_ode_runge_kutta_mgr (int task, void **state, …, 0)
  - void imsl_f_ode_runge_kutta (int neq, float *t, float tend, float y[], void *state, void fcn())
- **Required Arguments for imsl_f_ode_runge_kutta**
  - int neq (Input)
    - Number of differential equations
  - float *t (Input/Output)
    - Independent variable. On input, t is the initial independent variable value. On output, t is replaced by tend
  - float tend (Input)
    - Value of t at which the solution is desired
  - float y[] (Input/Output)
    - Array with neq components containing a vector of dependent variables.
  - void *state (Input/Output)
    - The current state of the ODE solution is held in a structure pointed to by state
  - void fcn (int neq, float t, float *y, float *yprime)
    - **User-supplied** function to evaluate the right-hand side

IMSL C Exercise 18
ODE Example



This example solves $\frac{dy}{dx} = -y$ over the interval [0, 1] with the initial condition y(0) = 1. The solution is $y(t) = e^{-t}$

```
#include <imsl.h>
#include <math.h>
void fcn (int neq, float t, float y[], float yprime[]);
main() {
  int neq = 1; /* Number of ode's */
  float t = 0.0; /* Initial time */
  float tend = 1.0; /* Final time */
  float y[1] = {1.0}; /* Initial condition */
  void *state;

  imsl_f_ode_runge_kutta_mgr (IMSL_ODE_INITIALIZE, &state, 0);
  imsl_f_ode_runge_kutta (neq, &t, tend, y, state, fcn);
  printf ("y[%f] = %f\n", t, y[0]);
  printf ("Error is: %e\n", exp( (double)(-tend) )-y[0]);
}
void fcn (int neq, float t, float y[], float yprime[]) {
  yprime[0] = -y[0];
}
```

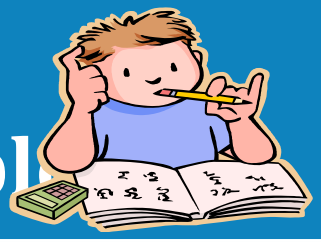
User-supplied function to evaluate the right-hand side

```
y[1.000000] = 0.367879
Error is: -9.149755e-09
```

IMSL Nonlinear Equations Example – Root of a Equations

- **zeros_sys_eqn**
  - Solves a system of n nonlinear equations f(x) = 0 using a modified Powell hybrid algorithm
- **Synopsis**
  - float *imsl_f_zeros_sys_eqn (void fcn(), int n, …, 0)
- **Required Arguments**
  - void fcn (int n, float x[], float f[]) (Input/Output)
    - User-supplied function to evaluate the system of equations to be solved, where n is the size of x and f, x is the point at which the functions are evaluated, and f contains the computed function values at the point x
  - int n (Input)
    - The number of equations to be solved and the number of unknowns
- **Return Value**
  - A pointer to the vector x that is a solution of the system of equations. To release this space, use free

IMSL C Exercise 19
Nonlinear Equations Exampl



The example solves 3 × 3 system of nonlinear equations

$$f_1(x) = x_1 + e^{x_1 - 1} + (x_2 + x_3)^2 - 27$$
$$f_2(x) = e^{x_2 - 2} / x_1 + x_3^2 - 10$$
$$f_3(x) = x_3 + \sin(x_2 - 2) + x_2^2 - 7$$

```
#include <imsl.h>
#include <stdio.h>
#include <math.h>
#define N 3
void fcn(int, float[], float[]);
void main() {
  float *x;
  float xguess[N] = {4.0, 4.0, 4.0};
  x = imsl_f_zeros_sys_eqn (fcn, N, IMSL_XGUESS, xguess, 0);
  imsl_f_write_matrix("The solution to the system is", 1, N, x, 0);
}
void fcn(int n, float x[], float f[]) {
  f[0] = x[0] + exp(x[0] - 1.0) + (x[1] + x[2]) * (x[1] + x[2]) - 27.0;
  f[1] = exp(x[1] - 2.0) / x[0] + x[2] * x[2] - 10.0;
  f[2] = x[2] + sin(x[1] - 2.0) + x[1] * x[1] - 7.0;
}
```

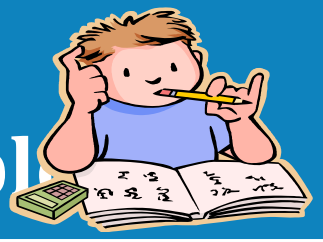
initial guess (4.0, 4.0, 4.0)

```
The solution to the system is
    1           2           3
    1           2           3
```

IMSL Optimization Example - Nonlinearly Constrained Minimization

- **constrained_nlp**
  - Solves a general nonlinear programming problem using a sequential equality constrained quadratic programming method
- **Synopsis**
  - float *imsl_f_constrained_nlp (void fcn(), int m, int meq, int n, int ibtype, float xlb[], float xub[], …, 0)
- **Required Arguments**
  - void fcn(int n, float x[], int iact, float *result, int *ierr) (Input)
    - User supplied function to evaluate the objective function and constraints at a given point
  - int m (Input)
    - Total number of constraints
  - int meq (Input)
    - Number of equality constraints
  - int n (Input)
    - Number of variables
  - int ibtype (Input)
    - Scalar indicating the types of bounds on variables
  - float xlb[] (Input, Output, or Input/Output)
    - Array with n components containing the lower bounds on the variables
    - (Input, if ibtype = 0; output, if ibtype = 1 or 2; Input/Output, if ibtype = 3)
    - If there is no lower bound on a variable, then the corresponding xlb value should be set to imsl_f_machine(8)
  - float xub[] (Input, Output, or Input/Output)
    - Array with n components containing the upper bounds on the variables
    - (Input, if ibtype = 0; output, if ibtype 1 or 2; Input/Output, if ibtype = 3)
    - If there is no upper bound on a variable, then the corresponding xub value should be set to imsl_f_machine(7)
- **Return Value**
  - A pointer to the solution x of the nonlinear programming problem. To release this space, use free. If no solution can be computed, then NULL is returned

IMSL C Exercise 20
Nonlinear Equations Example



$$\min F(x) = (x_1 - 2)^2 + (x_2 - 1)^2$$

subject to $g_1(x) = x_1 - 2x_2 + 1 = 0$

$g_2(x) = -x_1^2/4 - x_2^2 + 1 \geq 0$

2 Constraints
1 Equality constraints

```
include "imsl.h"
#define M 2
#define ME 1
#define N 2
void grad(int n, float x[], int iact, float result[]);
void fcn(int n, float x[], int iact, float *result, int *ierr);
void main() {
  int ibtype = 0;
  float *x, ans[2];
  static float xlb[N], xub[N];
  xlb[0] = xlb[1] = imsl_f_machine (8);
  xub[0] = xub[1] = imsl_f_constrained_nlp (fcn, M, ME, N, ibtype, xlb, xub, 0);
  x = imsl_f_write_matrix ("The solution is", 1, N, x, 0);
}
```

No lower and upper bound on a variable

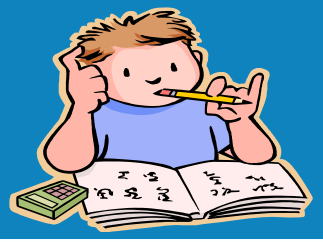
```
void fcn (int n, float x[], int iact, float *result, int *ierr) {
  float tmp1, tmp2;
  tmp1 = x[0] - 2.0e0;
  tmp2 = x[1] - 1.0e0;
  switch (iact) {
  case 0:
    *result = tmp1 * tmp1 + tmp2 * tmp2;
  break;
  case 1:
    *result = x[0] - 2.0e0 * x[1] + 1.0e0;
  break;
  case 2:
    *result = -(x[0]*x[0]) / 4.0e0 - x[1]*x[1] + 1.0e0;
  break;
  default: ;
  break;
}
  *ierr = 0;
  return;
}
```

```
The solution is
             1           2
        0.8229      0.9114
```

IMSL Regression Example - Multivariate Linear Regression

- **regression_prediction**
  - Computes predicted values, confidence intervals, and diagnostics after fitting a regression model
- **Synopsis**
  - float *imsls_f_regression_prediction (Imsls_f_regression *regression_info, int n_predict, float x[], ..., 0)
- **Required Argument**
  - Imsls_f_regression *regression_info (Input)
    - Pointer to a structure of type Imsls_f_regression containing information about the regression fit. See imsls_f_regression
  - int n_predict (Input)
    - Number of rows in x
  - float x[] (Input)
    - Array of size n_predict by the number of independent variables containing the combinations of independent variables in each row for which calculations are to be performed
- **Return Value**
  - Pointer to an internally allocated array of length n_predict containing the predicted values

IMSL C Exercise 20
Regression Prediction



$$y_i = \beta_0 + \beta_1 x_{i1} + \beta_2 x_{i2} + \ldots + \beta_k x_{ik} + \varepsilon_i \qquad i = 1, 2, \ldots, n$$

```
#include <imsls.h>
main() {
 #define INTERCEPT 1
 #define N_INDEPENDENT 4
 #define N_OBSERVATIONS 13
 #define N_COEFFICIENTS (INTERCEPT + N_INDEPENDENT)
 #define N_DEPENDENT 1
 float *y_hat, *coefficients;
 Imsls_f_regression *regression_info;
 float x[][N_INDEPENDENT] = {
 7.0, 26.0, 6.0, 60.0,
 1.0, 29.0, 15.0, 52.0,
 11.0, 56.0, 8.0, 20.0,
 11.0, 31.0, 8.0, 47.0,
 7.0, 52.0, 6.0, 33.0,
 11.0, 55.0, 9.0, 22.0,
 3.0, 71.0, 17.0, 6.0,
 1.0, 31.0, 22.0, 44.0,
 2.0, 54.0, 18.0, 22.0,
 21.0, 47.0, 4.0, 26.0,
 1.0, 40.0, 23.0, 34.0,
 11.0, 66.0, 9.0, 12.0,
 10.0, 68.0, 8.0, 12.0};
 float y[] = {78.5, 74.3, 104.3, 87.6, 95.9, 109.2, 102.7, 72.5, 93.1, 115.9, 83.8, 113.3, 109.4};
 coefficients = imsls_f_regression (N_OBSERVATIONS, N_INDEPENDENT, (float *)x, y, IMSLS_REGRESSION_INFO, &regression_info, 0);
 y_hat = imsls_f_regression_prediction (regression_info, N_OBSERVATIONS, (float*)x, 0);
 imsls_f_write_matrix("Predicted Responses", 1, N_OBSERVATIONS, y_hat, 0);
}
```

```
                    Predicted Responses
          1           2           3           4           5           6
       78.5        72.8       106.0        89.3        95.6       105.3

          7           8           9          10          11          12
      104.1        75.7        91.7       115.6        81.8       112.3

         13
      111.7
```

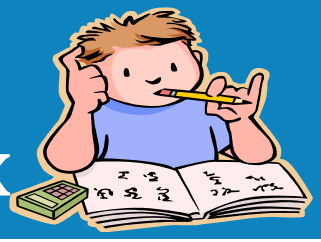
Fit the regression model
Return regression coefficients

Pointer to a structure of type imsls_f_regression containing information about the regression fit

IMSL Correlation and Covariance Example – Variance-Covariance Matrix

- **covariances**
  - Computes the sample variance-covariance or correlation matrix
- **Synopsis**
  - float *imsls_f_covariances (int n_rows, int n_variables, float x[], ...,0)
- **Required Arguments**
  - int n_rows (Input)
    - Number of rows in x
  - int n_variables (Input)
    - Number of variables
  - float x[] (Input)
    - Array of size n_rows × n_variables containing the data
- **Return Value**
  - If no optional arguments are used, imsls_f_covariances returns a pointer to an n_variables × n_variables array containing the sample variance-covariance matrix of the observations. The rows and columns of this array correspond to the columns of x

IMSL C Exercise 21
Variance-Covariance Matrix



```
#include <imsls.h>
#define N_VARIABLES 5
#define N_OBSERVATIONS 50
main() {
float *covariances, *means;
float x[] = {
1.0, 5.1, 3.5, 1.4, .2, 1.0, 4.9, 3.0, 1.4, .2,
1.0, 4.7, 3.2, 1.3, .2, 1.0, 4.6, 3.1, 1.5, .2,
1.0, 5.0, 3.6, 1.4, .2, 1.0, 5.4, 3.9, 1.7, .4,
1.0, 4.6, 3.4, 1.4, .3, 1.0, 5.0, 3.4, 1.5, .2,
1.0, 4.4, 2.9, 1.4, .2, 1.0, 4.9, 3.1, 1.5, .1,
1.0, 5.4, 3.7, 1.5, .2, 1.0, 4.8
1.0, 4.8, 3.0, 1.4, .1, 1.0, 4.3
1.0, 5.8, 4.0, 1.2, .2, 1.0, 5.7
1.0, 5.4, 3.9, 1.3, .4, 1.0, 5.1
1.0, 5.7, 3.8, 1.7, .3, 1.0, 5.1
1.0, 5.4, 3.4, 1.7, .2, 1.0, 5.1
1.0, 4.6, 3.6, 1.0, .2, 1.0, 5.1
1.0, 4.8, 3.4, 1.9, .2, 1.0, 5.0
1.0, 5.0, 3.4, 1.6, .4, 1.0, 5.1
1.0, 5.2, 3.4, 1.4, .2, 1.0, 4.7
1.0, 4.8, 3.1, 1.6, .2, 1.0, 5.4, 3.4, 1.5, .4,
1.0, 5.2, 4.1, 1.5, .1, 1.0, 5.5, 4.2, 1.4, .2,
1.0, 4.9, 3.1, 1.5, .2, 1.0, 5.0, 3.2, 1.2, .2,
1.0, 5.5, 3.5, 1.3, .2, 1.0, 4.9, 3.6, 1.4, .1,
1.0, 4.4, 3.0, 1.3, .2, 1.0, 5.1, 3.4, 1.5, .2,
1.0, 5.0, 3.5, 1.3, .3, 1.0, 4.5, 2.3, 1.3, .3,
1.0, 4.4, 3.2, 1.3, .2, 1.0, 5.0, 3.5, 1.6, .6,
1.0, 5.1, 3.8, 1.9, .4, 1.0, 4.8, 3.0, 1.4, .3,
1.0, 5.1, 3.8, 1.6, .2, 1.0, 4.6, 3.2, 1.4, .2,
1.0, 5.3, 3.7, 1.5, .2, 1.0, 5.0, 3.3, 1.4, .2};

covariances = imsls_f_covariances (N_OBSERVATIONS, N_VARIABLES, x, IMSLS_MISSING_VALUE_METHOD , 0, 0);
imsls_f_write_matrix ("The default case: variances/covariances", N_VARIABLES, N_VARIABLES, covariances, IMSLS_PRINT_UPPER, 0);
}
```

```
 The default case: variances/covariances
            1          2          3          4          5
1      0.0000     0.0000     0.0000     0.0000     0.0000
2                 0.1242     0.0992     0.0164     0.0103
3                            0.1437     0.0117     0.0093
4                                       0.0302     0.0061
5                                                  0.0111
```

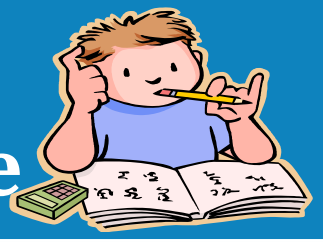
IMSLS_MISSING_VALUE_METHOD, int missing_value_method (Input)
Method used to exclude missing values in x from the computations

Visual Numerics

IMSL Correlation and Covariance Example – Pooled Variance-Covariance

- **pooled_covariances**
  - Compute a pooled variance-covariance from the observations
- **Synopsis**
  - float *imsls_f_pooled_covariances (int n_rows, int n_variables, float *x, int n_groups, ..., 0)
- **Required Argument**
  - int n_rows (Input)
    - Number of rows observations in the input matrix x
  - int n_variables (Input)
    - Number of variables to be used in computing the covariance matrix
  - float *x (Input)
    - A n_rows × n_variables + 1 matrix containing the data. The first n_variables columns correspond to the variables, and the last column (column n_variables must contain the group numbers)
  - int n_groups (Input)
    - Number of groups in the data
- **Return Value**
  - Matrix of size n_variables by n_variables containing the matrix of covariances
- **Optional Arguments**
  - IMSLS_IDO, int ido (Input)
    - Processing option

IMSL C Exercise 22
Pooled Variance-Covariance



The example computes a pooled variance-covariance matrix for the Fisher iris data. To illustrate the use of the ido argument, multiple calls to imsls_f_pooled_covariances are made

```
#include <stdio.h>
#include <stdlib.h>
#include <imsls.h>
main() {
  int nobs = 150;
  int nvar = 4;
  int n_groups = 3;
  int igrp = 0;
  static int ind[4] = {1, 2, 3, 4};
  int ifrq = -1;
  int iwt = -1;
  float *x, cov[16];
  float *means;
  int i;
  x = imsls_f_data_sets (3, 0);
  imsls_f_pooled_covariances (0, nvar, x, n_groups, IMSLS_IDO, 1, IMSLS_RETURN_USER, cov, IMSLS_X_INDICES, igrp, ind, ifrq, iwt, 0);
  for (i=0;i<15;i++) {
    imsls_f_pooled_covariances (10, nvar, (x+i*50), n_groups, IMSLS_IDO, 2, IMSLS_RETURN_USER, cov, IMSLS_X_INDICES, igrp, ind, ifrq, iwt, 0);
  }
  imsls_f_pooled_covariances (0, nvar, x, n_groups, IMSLS_IDO, 3, IMSLS_RETURN_USER, cov, IMSLS_X_INDICES, igrp, ind, ifrq, iwt,
        IMSLS_MEANS, &means, 0);
  imsls_f_write_matrix("Pooled Covariance Matrix", nvar, nvar, cov, 0);
  imsls_f_write_matrix("Means", n_groups, nvar, means, 0);
  free(means);
  free(x);
}
```

Fisher iris
150 observations and 5 variables

IDO = 1
First invocation with this data

IDO = 2
Intermediate invocation

IDO = 3
All statistics are updated for the n_rows observations

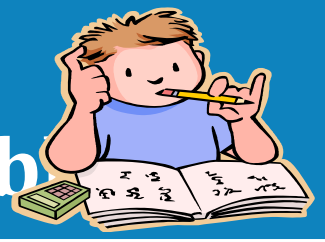
```
                Pooled Covariance Matrix
                1           2           3           4
1          0.2650      0.0927      0.1675      0.0384
2          0.0927      0.1154      0.0552      0.0327
3          0.1675      0.0552      0.1852      0.0427
4          0.0384      0.0327      0.0427      0.0419

                          Means
                1           2           3           4
1          5.006       3.428       1.462       0.246
2          5.936       2.770       4.260       1.326
3          6.588       2.974       5.552       2.026
```

Visual Numerics

IMSL Analysis of Variance and Designed Experiments Example – One-Way Classification Model

- **anova_oneway**
  - Analyzes a one-way classification model
- **Synopsis**
  - float imsls_f_anova_oneway (int n_groups, int n[], float y[], ..., 0)
- **Required Arguments**
  - int n_groups (Input)
    - Number of groups
  - int n[] (Input)
    - Array of length n_groups containing the number of responses for each group
  - float y[] (Input)
    - Array of length n [0] + n [1] + … + n [n_group -1] containing the responses for each group
- **Return Value**
  - The p-value for the F-statistic

IMSL C Exercise 23
Output One-Way ANOVA Tab



This example computes a one-way analysis of variance for data discussed by Searle (1971, Table 5.1, pp. 165~179). The responses are plant weights for six plants of three different types—three normal, two off-types, and one aberrant. The responses are given by type of plant in the right table.

Normal	Off-Type	Aberrant
101	84	32
105	88	
94		

```
#include <imsls.h>
main()  {
 int n_groups=3;
 int n[] = {3, 2, 1};
 float y[] = {101.0, 105.0, 94.0, 84.0, 88.0, 32.0};
 float p_value;
 float *anova_table;
 char *labels[] = {
 "degrees of freedom for among groups",  "degrees of freedom for within gro
 "total (corrected) degrees of freedom",  "sum of squares for among groups",
 "sum of squares for within groups",  "total (corrected) sum of squares",
 "among mean square",  "within mean square", "F-statistic",
 "p-value", "R-squared (in percent)",  "adjusted R-squared (in percent)",
 "est. standard deviation of within error",  "overall mean of y",  "coefficient of variation (in percent)"};
 p_value = imsls_f_anova_oneway (n_groups, n, y, IMSLS_ANOVA_TABLE, &anova_table, 0);
 imsls_f_write_matrix("* * * Analysis of Variance * * *\n", 15, 1, anova_table, IMSLS_ROW_LABELS, labels, IMSLS_WRITE_FORMAT, "%9.2f", 0);
}
```

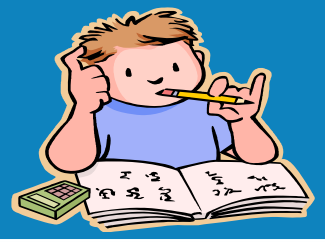
```
          * * * Analysis of Variance * * *

degrees of freedom for among groups            2.00
degrees of freedom for within groups           3.00
total (corrected) degrees of freedom           5.00
sum of squares for among groups             3480.00
sum of squares for within groups              70.00
total (corrected) sum of squares            3550.00
among mean square                           1740.00
within mean square                            23.33
F-statistic                                   74.57
p-value                                        0.00
R-squared (in percent)                        98.03
adjusted R-squared (in percent)               96.71
est. standard deviation of within error        4.83
overall mean of y                             84.00
coefficient of variation (in percent)          5.75
```

Visual Numerics

IMSL Nonparametric Statistics Example – Kruskal Wallis Test

- **kruskal_wallis_test**
  - Performs a Kruskal-Wallis test for identical population medians
- **Synopsis**
  - float *imsls_f_kruskal_wallis_test (int n_groups, int ni[], float y[], ..., 0)
- **Required Arguments**
  - int n_groups (Input)
    - Number of groups
  - int ni[] (Input)
    - Array of length n_groups containing the number of responses for each of the n_groups groups
  - float y[] (Input)
    - Array of length ni[0] + ... + ni[n_groups-1] that contains the responses for each of the n_groups groups. y must be sorted by group, with the ni[0] observations in group 1 coming first, the ni[1] observations in group two coming second, and so on
- **Return Value**
  - Array of length 4 containing the Kruskal-Wallis statistics
    - I stat[I]
    - 0 Kruskal-Wallis H statistic
    - 1 Asymptotic probability of a larger H under the null hypothesis of identical population medians
    - 2 H corrected for ties
    - 3 Asymptotic probability of a larger H (corrected for ties) under the null hypothesis of identical populations
- **Optional Arguments**
  - IMSLS_FUZZ, float fuzz (Input)
    - Constant used to determine ties in y. If (after sorting) $|y[i] - y[i+1]|$ is less than or equal to fuzz, then a tie is counted
  - IMSLS_RETURN_USER, float stat[] (Output)
    - User defined array for storage of Kruskal-Wallis statistics

IMSL C Exercise 24
Kruskal Wallis Test



The example is taken from Conover (1980, page 231). The data represents the yields per acre of four different methods for raising corn. Since H = 25.5, the four methods are clearly different. The warning error is always printed when the Beta approximation is used, unless printing for warning errors is turned off

```
#include <imsls.h>
void main()  {
  int ngroup = 4, ni[] = {9, 10, 7, 8};
  float y[] = {83., 91., 94., 89., 89., 96., 91., 92., 90., 91., 90.,
  81., 83., 84., 83., 88., 91., 89., 84., 101., 100., 91.,
  93., 96., 95., 94., 78., 82., 81., 77., 79., 81., 80., 81.};
  float fuzz = .001, stat[4];
  char *rlabel[] = {"H (no ties) =",   "Prob (no ties) =", "H (ties) =", "Prob (ties) ="};
  imsls_f_kruskal_wallis_test (ngroup, ni, y, IMSLS_FUZZ, fuzz, IMSLS_RETURN_USER, stat, 0);
  imsls_f_write_matrix(" ", 4, 1, stat, IMSLS_ROW_LABELS, rlabel, 0);
}
```

```
*** WARNING ERROR from imsls_kruskal_wallis_test. The chi-squared degrees
*** of freedom are less than 5, so the Beta approximation is used.
H (no ties)           =          25.46
Prob (no ties)        =          0.00
H (ties)              =          25.63
Prob (ties)           =          0.00
```

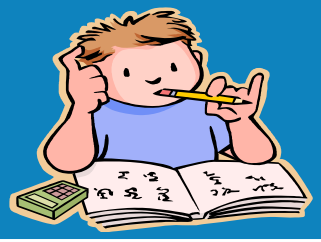
IMSL Multivariate Analysis Example – K-Means Cluster Analysis

- **cluster_k_means**
  - Performs a K-means (centroid) cluster analysis
- **Synopsis**
  - int *imsls_f_cluster_k_means (int n_observations, int n_variables, float x[], int n_clusters, float cluster_seeds, ..., 0)
- **Required Arguments**
  - int n_observations (Input)
    - Number of observations
  - int n_variables (Input)
    - Number of variables to be used in computing the metric
  - float x[] (Input)
    - Array of length n_observations × n_variables containing the observations to be clustered
  - int n_clusters (Input)
    - Number of clusters
  - float cluster_seeds[] (Input)
    - Array of length n_clusters × n_variables containing the cluster seeds, i.e., estimates for the cluster centers
- **Return Value**
  - The cluster membership for each observation is returned

Visual Numerics

www.vni.com.tw

IMSL C Exercise 25
K-Means Cluster Analysis



This example performs K-means cluster analysis on Fisher's iris data, which is obtained by function imsls_f_data_sets. The initial cluster seed for each iris type is an observation known to be in the iris type

```
#include <stdio.h>
#include <imsls.h>
main() {
 #define N_OBSERVATIONS 150
 #define N_VARIABLES 4
 #define N_CLUSTERS 3
 float x[N_OBSERVATIONS][5];
 float cluster_seeds[N_CLUSTERS][N_VARIABLES];
 float cluster_means[N_CLUSTERS][N_VARIABLES];
 float cluster_ssq[N_CLUSTERS];
 int cluster_variables[N_VARIABLES] = {1, 2, 3, 4};
 int cluster_counts[N_CLUSTERS];
 int cluster_group[N_OBSERVATIONS];
 int i;
 imsls_f_data_sets (3, IMSLS_RETURN_USER, x, 0);
 for (i=0; i<N_VARIABLES; i++) {
 cluster_seeds[0][i] = x[0][i+1];
 cluster_seeds[1][i] = x[50][i+1];
 cluster_seeds[2][i] = x[100][i+1];
 }
 imsls_f_cluster_k_means (N_OBSERVATIONS, N_VARIABLES, (float*)x, N_CLUSTERS, (float*)cluster_seeds,
       IMSLS_X_COL_DIM, 5, IMSLS_CLUSTER_VARIABLE_COLUMNS, cluster_variables, IMSLS_CLUSTER_COUNTS_USER,
       cluster_counts, IMSLS_CLUSTER_MEANS_USER, cluster_means, IMSLS_CLUSTER_SSQ_USER, cluster_ssq,
       IMSLS_RETURN_USER, cluster_group, 0);
 imsls_f_write_matrix("Cluster Means", N_CLUSTERS, N_VARIABLES, (float*)cluster_means, 0);
 imsls_f_write_matrix("Cluster Sum of Squares", 1, N_CLUSTERS, cluster_ssq, 0);
 imsls_i_write_matrix("# Observations in Each Cluster", 1, N_CLUSTERS, cluster_counts, 0);
}
```

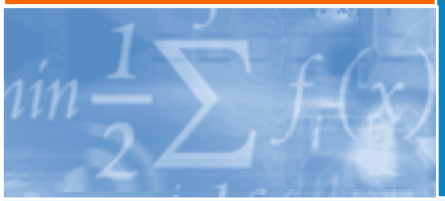
Fisher iris
150 observations and 5 variables

4 Variables
150 Observations
3 Clusters

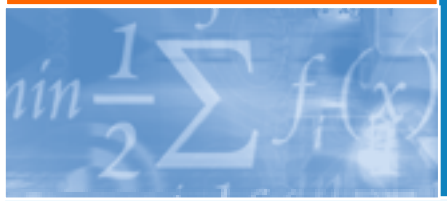
```
                    Cluster Means
             1           2           3           4
1        5.006       3.428       1.462       0.246
2        5.902       2.748       4.394       1.434
3        6.850       3.074       5.742       2.071

      Cluster Sum of Squares
         1           2           3
     15.15       39.82       23.88

# Observations in Each Cluster
          1    2    3
         50   62   38
```

IMSL C Numerical Library
Useful Information

- Calling IMSL CNL from VB or Excel
  - http://www.vni.com/products/imsl/c/visualbasic.html
- Calling IMSL Fortran Library from C Language
  - http://www.vni.com/tech/imsl/8902.pdf
- Calling IMSL CNL for MVS version from Borland C++ Builder
  - http://www.vni.com/tech/imsl/bcc55_cnldll.zip
- Calling IMSL CNL from C Language
  - http://www.vni.com.tw/tw/tech/imsl/



Open Discussion



Thanks!

Appendix

IMSL Matrix Storage Modes

IMSL CNL
Matrix Storage Modes 1/10

- IMSL CNL functions require input consisting of matrix dimension values and all values for the matrix entries
- The values are stored in row-major order in the arrays
- Each function processes the input array and typically returns a pointer to a “result”

IMSL CNL
Matrix Storage Modes 2/10

- General Mode
  - n x n matrix
  - float, double, f_complex, d_complex
- Rectangular Mode
  - m x n matrix
  - float, double, f_complex, d_complex
- Symmetric Mode
  - $A^T = A$
  - n x n matrix
  - float, double

IMSL CNL
Matrix Storage Modes 3/10

- Hermitian Mode
  - n x n matrix $A^H = \overline{A}^T = A$ ($\overline{A}^T$ is the complex conjugate of A)
  - f_complex, d_complex

IMSL CNL
Matrix Storage Modes 4/10

- Sparse Coordinate Storage Format
  - Only the nonzero of a sparse matrix need to be communicated to a function

```
typedef struct{
        int row;
        int col;
        float val;
} Imsl_f_sparse_elem:

typedef struct {
        int row;
        int col;
        f_complex val;
} Imsl_c_sparse_elem;
```

IMSL CNL
Matrix Storage Modes 5/10

```
#include <imsl.h>
main()  {
Imsl_f_sparse_elem a[] = {
{0, 0, 2.0},
{1, 1, 9.0},
{1, 2, -3.0},
{1, 3, -1.0},
{2, 2, 5.0},
…. };
```

$$A=\begin{bmatrix} 2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 9 & -3 & -1 & 0 & 0 \\ 0 & 0 & 5 & 0 & 0 & 0 \\ -2 & 0 & 0 & -7 & -1 & 0 \\ -1 & 0 & 0 & -5 & 1 & -3 \\ -1 & -2 & 0 & 0 & 0 & 6 \end{bmatrix}$$

row	0	1	1	1	2	3	3	3	4	4	4	4	5	5	5
col	0	1	2	3	2	0	3	4	0	3	4	5	0	1	5
Val	2	9	-3	-1	5	-2	-7	-1	-1	-5	1	-3	-1	-2	6

```
Imsl_f_sparse_elem b[15];
b[0].row = b[0].col = 0;        b[0].val = 2.0;
b[1].row = b[1].col = 1;        b[l].val = 9.0;
b[2].row = 1; b[2].col = 2;     b[2].val = -3.0;
b[3].row = 1; b[3].col = 3;     b[3].val = -1.0;
b[4].row = b[4].col = 2;        b[4].val = 5.0;
…..
};
```

IMSL CNL
Matrix Storage Modes 6/10

- Band Storage Format
  - M x N matrix
  - All of nonzero elements close to the main diagonal
  - $A_{ij} = 0$ if i-j < nlca or j-i > nuca
  - M = nlca + nuca + 1
  - nlca lower codiagonals, nuca upper codiagonals are stored in the rows of an array of size m x N
  - The elements are stored in the same column of the array as type are in the matrix

IMSL CNL
Matrix Storage Modes 7/10

$$A=\begin{bmatrix} A_{0,0} & A_{0,1} & A_{0,2} & 0 & 0 \\ A_{1,0} & A_{1,1} & A_{1,2} & A_{1,3} & 0 \\ 0 & A_{2,1} & A_{2,2} & A_{2,3} & A_{2,4} \\ 0 & 0 & A_{3,2} & A_{3,3} & A_{3,4} \\ 0 & 0 & 0 & A_{4,3} & A_{4,4} \end{bmatrix}$$

data be arranged as

$$\begin{bmatrix} 0 & 0 & A_{0,2} & A_{1,3} & A_{2,4} \\ 0 & A_{0,1} & A_{1,2} & A_{2,3} & A_{3,4} \\ A_{0,0} & A_{1,1} & A_{2,2} & A_{3,3} & A_{4,4} \\ A_{1,0} & A_{2,1} & A_{3,2} & A_{4.3} & 0 \end{bmatrix}$$

$$A=\begin{bmatrix} 10 & 1 & 0 & 0 & 0 \\ 5 & 20 & 2 & 0 & 0 \\ 0 & 6 & 30 & 3 & 0 \\ 0 & 0 & 7 & 40 & 4 \\ 0 & 0 & 0 & 8 & 50 \end{bmatrix}$$

declaration

```
float  a[] = {0.0, 1.0, 2.0, 3.0,
4.0, 10.0, 20.0, 30.0, 40.0, 50.0,
5.0, 6.0, 7.0, 8.0, 0.0};
```

IMSL CNL
Matrix Storage Modes 8/10

- Compressed Sparse Column (CSC) format
  - The Harwell-Boeing Sparse Matrix Collection is a set of standard test matrices arising from problems in linear systems, least squares, and eigenvalue calculations
- The scheme is column oriented, with each column held as a sparse vector
- Data for each column are stored consecutively and in order

IMSL CNL
Matrix Storage Modes 9/10

- colptr: location of first entry
- rowind: rwo indices
- values: nonzero entries

$$\begin{bmatrix} 1 & -3 & 0 & -1 & 0 \\ 0 & 0 & -2 & 0 & 3 \\ 2 & 0 & 0 & 0 & 0 \\ 0 & 4 & 0 & -4 & 0 \\ 5 & 0 & -5 & 0 & 6 \end{bmatrix}$$

Subscripts	0	1	2	3	4	5	6	7	8	9	10
colptr	0	3	5	7	9	11					
rowind	0	4	2	3	0	1	4	0	3	4	1
values	1	5	2	4	-3	-2	-5	-1	-4	6	3

IMSL CNL
Matrix Storage Modes 10/10

- Linear Systems
  - Linear equations with full matrices
    - lin_sol_gen
  - Linear equations with band matrices
    - lin_sol_gen_band
  - Linear equations with general sparse matrices
    - lin_sol_gen_coordinate
  - :

IMSL C on hpc.ustc.edu.cn

- telnet to 202.38.64.91
- Account: train
- Password: imsl
- IMSL installed path: /opt/vni