MIST を勧める５つの理由

高　橋　友　和

Ⅰ．はじめに

MIST は，C++ で書かれた音声・画像処理のためのソフトウェアライブラリであり，情報科学研究科の学生を中心に構成された MIST プロジェクトチームにより現在も開発が続けられている。MIST のライブラリパッケージはプロジェクトの Web サイト http://mist.suenaga.m.is.nagoya-u.ac.jp からダウンロードできる。また，同サイトにて使用方法やプロジェクトの活動の概要なども閲覧できる。

私は MIST プロジェクトチームのメンバの一人である。この文章の目的は MIST をたくさんの人に知ってもらい，そして実際に利用してもらうことである。この文章のタイトルは MIST を勧める 5 つの理由とした。私自身，MIST を利用し始めてから研究で必要な実験のプログラムを書くのがたいへん楽になったし，私が所属している研究室の学生も皆 MIST を利用しており，そこでも MIST はたいへん好評のようである。MIST を今まで知らなかったという人はこの文章を読んで是非一度 MIST を試してみて欲しい。同じ大学内で便利なライブラリが開発されているのにそれを知らないのはもったいないと思う。MIST をすでに利用しているという人は是非周りの人にも MIST を紹介してもらいたい。その際にこの文章が役立てば嬉しく思う。MIST プロジェクトの目的はライブラリの開発と公開であり，公開するからにはたくさんの人に利用してもらいたいと思っている。今後のさらなるライブラリの質的向上のためにもユーザからのフィードバックが重要である。

この後に続くⅡ節では MIST プロジェクトとその活動の概要について紹介する。Ⅲ節では MIST を勧める 5 つの理由を一つずつ説明する。MIST ライブラリの機能についてはⅣ節で紹介する。とにかく MIST を実際に使用して感触を確かめたいという人のために，Ⅴ節で MIST の入手，環境設定，サンプルプログラムを用いた簡単な利用方法，コンパイル方法を詳しく解説する。最後のⅥ節には MIST プロジェクトの今後の展望を書く。

Ⅱ．MIST プロジェクト

MIST の正式名称は Media Integration Standard Toolkit である。MIST プロジェクトは名古屋大学情報系 COE の若手横断プロジェクトの 1 つであり，今年で 4 年目を迎えるプロジェクトである。名古屋大学情報系 COE は「社会情報基盤のための音声・映像の知的統合（Intelligent Media Integration for Social Information Infrastructure）」を研究の目的としており，MIST の名前はこの一部をとって付けた。若手横断プロジェクトは，分野をまたがった若手研究者の視野

拡大と研究交流をねらったプロジェクトで，複数の異なる研究室の助手やポスドク，博士後期課程の学生がチームを組んでプロジェクト計画の申請を行うものである。MIST プロジェクト 1 年目は情報科学研究科の 3 つの研究室（村瀬研，末永研，大西研）から筆者を含め 1 人ずつ計 3 人のチームで活動する小さなプロジェクトであったが，現在は 6 人のチームメンバと学生，教官からなる多数の支援者の協力を受けて活動を続けている。

2005 年 12 月より MIST プロジェクトの一般公開を開始した。このことはプロジェクト発足当時からの念願であった。プロジェクトの Web サイトでは定期的にリリースされる安定版ライブラリパッケージのダウンロードの他，ライブラリドキュメント，使用方法や環境設定などに関するチュートリアル，MIST プロジェクトの活動の概要などが閲覧できる。まだ日本語サイトしか用意されていないため，日本の大学や企業からのアクセスがほとんどであるが，MIST は実際多くの人に利用されている。一般公開後，2006 年 8 月 11 日現在までの Web サイトアクセス数は 40,000 を超え，公開後にリリースされたライブラリパッケージのダウンロード数は 1,300 を超えている。ただし，ライブラリのダウンロード方法は Web サイトからの安定板パッケージダウンロードと Subversion と呼ばれるソフトウェアを利用した方法の 2 とおりあり，実質的なライブラリのダウンロード数は，上記の 2 倍程度あると思われる。また，ユーザは Web 上で簡単にバグ報告を行うことができ，バグフィクスの結果も容易に閲覧できる（8 月 11 日現在，サイトへのスパム的な書き込みが激増し対応が困難となったため，一時的に Web 上でのバグ報告機能を停止）。

Ⅲ．MIST を勧める 5 つの理由

MIST を勧める 5 つの理由を 4W1H（Whoever, Wherever, Whenever, Whatever, However）の形式で以下に挙げ，説明する。

理由 1：誰でも利用できる（Whoever）

MIST の導入方法は非常にシンプルで，特別な知識を必要としない。さらに Web サイトにはライブラリの入手から環境設定，使用方法を詳しく解説するチュートリアルが用意されており，今までほかのライブラリを導入しようとしたけれど設定が難しくて断念した経験のある人でも迷うことなく利用を開始できる。実際の研究の現場で培われてきた多数の強力なアルゴリズムが用意されており，ソースコードも公開されているため，とにかく豊富な機能と速度が欲しいという人はもちろん，プログラミングやアルゴリズムを学びたいという人にもお勧めできる。

理由 2：どこでも動作する（Wherever）

研究室の計算機環境は，Windows か Linux が大半を占めていると考える。MIST は VC7（MS Visual Studio .NET2003）以降，及び gcc3.0 以降のコンパイラに対応しており，開発環境を気にせずにプログラミングができる（詳細は Web サイトの対応コンパイラ表参照）。

理由 3：いつでも最新版が手に入る（Whenever）

定期的にリリースされる MIST の安定版パッケージはプロジェクトの Web サイトからいつでもダウンロード可能である。また，ライブラリの使用方法を解説するドキュメント，チュートリアルなども同 Web サイトにていつでも閲覧可能である。Web サイトにはバグ報告機能も用意されており，リリースされたライブラリパッケージに万が一バグが見つかり，それが修正された場合，その修正が反映された最新版のライブラリは Subversion 経由でいつでもダウンロードすることができる（Subversion 経由でのダウンロード方法の詳細は Web サイトを参照）。また，MIST は名古屋大学内で開発されており，その気になればいつでも開発者に会って直接相談することもできる。

理由 4：何でも扱える（Whatever）

MIST のコンテナは C++ のテンプレートというデータ型を抽象化する機能を用いて実現されているため，コンテナの要素の型として整数型や浮動小数点型などの既存の型はもちろんユーザ定義型でも何でも使用可能である。MIST のアルゴリズムも同じくテンプレート機能と関数の多重定義機能を用いて実装されており，同じ関数名で異なる種類のコンテナの処理ができるため，ユーザはコンテナの種類やコンテナの要素の型の違いによって関数を呼び分けたり，たくさんの関数名を覚えたりする必要がない。

理由 5：どんな目的でも利用できる（However）

MIST のライセンスは，一般的なオープンソースソフトウェアのライセンスの中で比較的制限の緩い BSD ライセンスに準拠している。GPL（GNU Public License）などの他のライセンスを採用しているライブラリを使用する場合，それを使用して作成したソフトウェアを配布する際にソースコードを公開しなければならないといった制限を受ける場合がある。近年，研究室が企業などと共同研究を行いながら製品や商用ソフトウェアを開発することも多くなってきており，そのような場合にはこの制限は致命的である。これに対して MIST のライセンスにはこのような制限が存在しないため，どんな目的でも利用しやすい（詳細は Web サイトのライセンスページ参照）。

Ⅳ．MIST の機能

MIST は，大きく分けてコンテナ，アルゴリズム，ファイル入出力の 3 つのモジュールから構成されている。MIST を使った音声・画像処理の基本的な流れはつぎのようになる。(1) ファイルからデータを読み出してコンテナに格納する。(2) アルゴリズムはデータを格納したコンテナを入力として受け取り，処理結果として得られたデータをコンテナに格納して出力として返す。(3) コンテナに格納されたデータをファイルに書き出す。以降でそれら 3 つのモジュールについて説明する。

1　コンテナ

同じ型のデータをたくさん集めて入れておくための入れ物，すなわち配列や線形リストなどのデータ構造を総称してコンテナと呼ぶ。C++ の標準ライブラリ STL (Standard Template Library）には，vector や list といったコンテナクラスが用意されている。MIST にもいくつかの有用なコンテナクラスが用意されている。これらの多くはアルゴリズムの入出力やファイル入出力の対象となるデータを格納するために用いられる。ここではそのうちの 5 つの基本的なコンテナクラスを紹介する。array コンテナは，つぎに紹介する 4 つのクラスの基底クラスで，STL の vector に似たサイズ可変の 1 次元配列である。array1，array2，array3 コンテナはそれぞれ 1 次元配列，2 次元配列，3 次元配列である。matrix コンテナ は array2 コンテナと同様 2 次元配列の形状をしているが，こちらは数値計算で用いられる行列の要素を格納するためのものであり，行列同士の足し算，引き算，内積，行列の定数倍，転置などが定義されている。

2　アルゴリズム

MIST にはたくさんの強力なアルゴリズムが用意されており，それらのアルゴリズムの多くはデータを格納したコンテナを入出力とした関数の形で定義されている。ここではその一部を紹介する。

画像処理系　線形フィルタ，メディアンフィルタ，領域拡張法，画像補間，インターレス除去，閾値選択，モルフォロジー演算，ラベリング，図形分割，境界画素抽出，最頻値フィルタ，細線化，スケルトン抽出，距離変換，ボロノイ分割，図形融合，非剛体レジストレーション，ボリュームレンダリング，等濃度面表示，カメラキャリブレーション。

信号処理系　高速フーリエ変換 (FFT)，離散コサイン変換（DCT)，離散サイン変換 (DST)。

統計処理系　平均値計算，分散計算，ヒストグラム作成，混合正規分布の推定。

行列･数値計算系　トレース, 行列式, 逆行列, 線形連立方程式の導解, QR 分解, LU 分解, 固有値･固有ベクトルの導出，特異値分解。

実装補助系　型情報の取得, スマートポインタ（自動的にメモリ管理を行うポインタ), 時間計測，擬似乱数生成，計算機環境情報取得，マルチスレッド，シングルトン，ハッシュ関数，関数の最小化，自由曲線・曲面。

3　ファイル入出力

データをファイルから読み出してコンテナに格納したり，コンテナに格納されたデータをファイルに書き出したりする。MIST は以下に示す多彩なファイルフォーマットに対応している。

画像フォーマット　RAW，BMP，PNM，JPEG，PNG，GIF，TIFF，TGA，DICOM

音声フォーマット　WAV

動画像フォーマット　MPEG（array2 コンテナを介したフレーム画像の読み書き）

V．MIST を使ってみよう

この節では，MIST の入手と環境設定の方法，簡単なサンプルプログラムを使った MIST の基本的な使用方法を順に解説する。Web サイトのチュートリアルにはより詳しい解説があるので，そちらも併せて参照すると理解がより深まる。

1　ライブラリの入手・環境設定

MIST プロジェクトの Web サイト http://mist.suenaga.m.is.nagoya-u.ac.jp のトップページ → 左側のダウンロードへ移動 → 最新のスナップショット Version 1.3.3（8 月 29 日現在）を選択 → ダウンロードしたファイルを適当な場所（この解説では“hogehoge/”とする）に展開。

VS.NET2003（Windows）の環境設定

(1) VS.NET を起動 → 上段メニューからファイル → 新規作成 → プロジェクト → Visual C++ プロジェクト → 空のプロジェクト → 適当なプロジェクト名（この解説では“hoge”とする）をつけて OK。
(2) ソリューションエクスプローラからソースファイルのフォルダ右クリック → 追加 → 新しい項目の追加 → 適当なファイル名（この解説では“hoge.cpp”とする）をつけて開く。
(3) 上段メニューからプロジェクト →“hoge”のプロパティ → C/C++ → 全般 → 追加のインクルードディレクトリに MIST を展開したディレクトリパス“hogehoge/mist-v1.3.3”を追加。
(4) サンプルプログラム 1 を先ほど作成したファイル“hoge.cpp”に記述。Ctl+F5 でビルド & デバッグなしを実行し，ビルドエラーが出なければ環境設定は正常に行われている。

gcc3.0 以上（Linux）の環境設定

(1) 適当なファイル（この解説では“hoge.cpp”とする）にサンプルプログラム 1 を記述する。
(2) 以下のようにコンパイルして，コンパイルエラーが出なければ環境設定は正常に行われている。

```
% g++ hoge.cpp -Wall -DNDEBUG -O2 -Ihogehoge/mist-v1.3.3 -lm -lpthread
```

```
1 #include <mist/mist.h>
2
3 int main( void )
4 {
5       return 0;
6 }
```

サンプルプログラム 1

2　画像ファイルの入出力

ここからは bmp 画像を扱うサンプルプログラムをいくつか解説する。カラー画像でもグレースケール画像でも良いので“image.bmp”という名前の BMP ファイルを用意して，cpp ファイルが存在するディレクトリと同じディレクトリに置いて欲しい。

初めにファイルの入出力について解説する。サンプルプログラム 2 － 1 を実行すると“output.bmp”という名前のグレースケール画像が作成される。以降，サンプルプログラムの各行の先頭には解説の際に参照しやすいように行番号をつけることにする。サンプルプログラム 2 － 1 を先頭から順に解説していく。01 行では MIST のコンテナが定義されているヘッダファイルをインクルードしている。MIST のコンテナを用いる場合には必ずこの行の記述が必要である。02 行では BMP ファイル入出力のためのヘッダファイルをインクルードしている。サンプルプログラム 2 － 1 の中で一見して難しそうなところは 06 行である。この行では“unsigned char”型の要素を格納するための“array2”コンテナ“img”を宣言している。“mist::”は後に続くクラス名や関数名が“mist”という名前空間の中に定義されていることをあらわす。“array2”は，MIST の 2 次元配列用コンテナである。その後に続く型名を“<”，“>”で囲む見慣れない表現は C++ のテンプレート機能を用いる際に使用されるものであり，この場合“unsigned char”は配列に格納される要素の型をあらわしている。続く 07 行で“image.bmp”の画像サイズ（画素数）に合わせて“img”のメモリが自動的に確保され，各画素の値が“img”の各要素として格納される。このとき，“image.bmp”がカラー画像であった場合には，画素値が自動的に 8bit（“unsigned char”）グレースケールに変換される。最後に 08 行で“img”の中身が“output.bmp”に出力される。

```
01 #include <mist/mist.h>
02 #include <mist/io/bmp.h>
03
04 int main( void )
05 {
06   mist::array2< unsigned char > img;
07   mist::read_bmp( img, "image.bmp" );
08   mist::write_bmp( img, "output.bmp" );
09
10   return 0;
11 }
```

サンプルプログラム 2 － 1

カラー画像をそのまま入力して扱いたい場合には，サンプルプログラム 2 － 2 のように記述すれば良い。サンプルプログラム 2 － 1 から変化したところは 06 行である。“rgb”も MIST の

コンテナの一種で，これはRGB画素を扱うためのものである。このようにコンテナを入れ子状に使用することもできる。

```
01 #include <mist/mist.h>
02 #include <mist/io/bmp.h>
03
04 int main( void )
05 {
06   mist::array2< mist::rgb< unsigned char > > img;
07   mist::read_bmp( img, "image.bmp" );
08   mist::write_bmp( img, "output.bmp" );
09
10   return 0;
11 }
```

サンプルプログラム２－２

３　コンテナ操作

コンテナ内の任意の要素にアクセスする方法について，画像の2値化処理を例にとって解説する。サンプルプログラム３－１を実行すると“binary.bmp”という2値画像が作成される。前回から変化したところは08行から14行である。この部分では2次元配列の各要素が2重ループを用いて処理されている。気を付けるべき点は08行，10行の各ループの終了条件に“img”のメンバ関数“height( )”，“width( )”がそれぞれ使われているところである。これらのメンバ関数によって2次元配列の高さと幅の情報を取得することができる。12行に示すように“img”の任意の要素へのアクセスは“( )”演算子を用いて，“img(i, j)”のように直観的な表現で書ける。

```
00 #include <mist/mist.h>
01 #include <mist/io/bmp.h>
02
03 int main( void )
04 {
05   mist::array2< unsigned char > img;
06   mist::read_bmp( img, "image.bmp" );
07
08   for( size_t j = 0 ; j < img.height( ) ; j ++ )
09   {
```

```
10          for( size_t i = 0 ; i < img.width( ) ; i ++ )
11          {
12                  img( i, j ) = ( img( i, j ) >= 128 ) ? 255 : 0;
13          }
14   }
15   mist::write_bmp( img, "binary.bmp" );
16
17   return 0;
18 }
```

サンプルプログラム３－１

カラー画像の画素のR値，G値，B値にアクセスする方法を，サンプルプログラム３－２を用いて解説する。プログラムを実行すると“octonary.bmp”という画像が作成される。R，G，B値それぞれを２値化しているので出力画像は８（$= 2^3$）値画像となる。13行から15行に示すように各R，G，B値には，メンバ変数“r”，“g”，“b”を用いて簡単にアクセスできる。

```
01 #include <mist/mist.h>
02 #include <mist/io/bmp.h>
03
04 int main( void )
05 {
06   mist::array2< mist::rgb< unsigned char > > img;
07   mist::read_bmp( img, "image.bmp" );
08
09   for( size_t j = 0 ; j < img.height( ) ; j ++ )
10   {
11          for( size_t i = 0 ; i < img.width( ) ; i ++ )
12          {
13                  img( i, j ).r = ( img( i, j ).r >=128 ) ? 255 : 0;
14                  img( i, j ).g = ( img( i, j ).g >=128 ) ? 255 : 0;
15                  img( i, j ).b = ( img( i, j ).b >=128 ) ? 255 : 0;
16          }
17   }
18   mist::write_bmp( img, "octonary.bmp" );
19
20   return 0;
21 }
```

サンプルプログラム３－２

STL を知っている人はサンプルプログラム 3 － 3 を見て欲しい。実行結果はサンプルプログラム 3 － 1 と同様である。04 行から 14 行に“thresholding”というファンクタクラスを定義しており，21 行で“for_each”という STL のアルゴリズムを用いて“img”内の全要素にそのファンクタを適用している。このように MIST のコンテナは STL のコンテナに準拠するように設計されており，STL の強力なアルゴリズムをそのまま利用することもできる。

```
01 #include <mist/mist.h>
02 #include <mist/io/bmp.h>
03
04 struct thresholding
05 {
06   unsigned char th_;
07   thresholding( const unsigned char th = 0 ) : th_( th )
08   {
09   }
10   void operator( )( unsigned char &v ) const
11   {
12         v = ( v >= th_ ) ? 255 : 0;
13   }
14 };
15
16 int main( void )
17 {
18   mist::array2< unsigned char > img;
19   mist::read_bmp( img, "image.bmp" );
20   std::for_each( img.begin( ), img.end( ), thresholding( 128 ) );
21   mist::write_bmp( img, "binary.bmp" );
22
23   return 0;
24 }
```

サンプルプログラム３－３

4　画像処理アルゴリズムの適用

最後にもう少しだけ画像処理らしい例を挙げて，MIST のアルゴリズム群の一部を解説する。その他のアルゴリズムについてはⅣ節 2，ならびに Web サイトを参照して欲しい。サンプルプログラム 4 では，まず入力されたグレースケール画像が「大津の判別分析法」により決定された適切な閾値によって 2 値化される。つぎに opening 演算によりノイズ除去された後，図形の連

結成分（白画素の塊）のラベリング処理がなされる。最後にラベル付けされた各画素はラベルごとにランダムに割り振った色で塗り分けられ，その画像“labeling.bmp”と連結成分数が出力される。

14 行は“threshold”という関数で“img”に対する閾値“th”を決定している。この関数を利用するためには 03 行の記述が必要である。24 行では，2 値画像に対する半径“1.0”の円を構造要素とする“opening”演算が行われている。この演算はモルフォルジ演算と呼ばれており，04 行のヘッダファイル中に定義されている。詳細な解説は画像処理の専門書に譲るが，イメージとしては図形の外周を円で削り，その後削ったところに円をはめ込むような処理であり，画像処理では雑音・小成分除去などによく用いられる。29 行はラベリング処理で図形の各連結成分（白画素の塊）に異なるラベル番号を付ける処理であり，この機能を利用するためには 05 行をインクルードする必要がある。27 行で宣言した“label_num”にラベルの数（連結成分の数）が返り，28 行で宣言した“label”には，各画素のラベル番号を要素として持つ 2 次元配列が返る。31 行から 38 行までは，最終的な出力画像をより見やすくするためにラベル番号にランダムな色を割り当てるカラーテーブル“color_tab”を作成する処理である。この中で 06 行のヘッダファイルに定義された擬似乱数生成が用いられている。31 行の“std::clock（）”は乱数生成器を初期化するために用いられる。40 行から 45 行では，実際に“label”と“color_tab”を参照しながら最終的には出力画像“out”をファイルに書き出し，標準出力に連結成分数を表示する。

```
01 #include <mist/mist.h>
02 #include <mist/io/bmp.h>
03 #include <mist/threshold.h>
04 #include <mist/filter/morphology.h>
05 #include <mist/filter/labeling.h>
06 #include <mist/random.h>
07 #include <ctime>
08
09 int main( void )
10 {
11   mist::array2< unsigned char > img;
12   mist::read_bmp( img, "image.bmp" );
13
14   const unsigned char th = mist::discriminant_analysis::threshold(
     img );
15   for( size_t i = 0 ; i < img.size( ) ; i ++ )
16   {
17         img[ i ] = ( img[ i ] >= th ) ? 255 : 0;
18   }
19   mist::write_bmp( img, "binary.bmp" );
20
```

```
23
24   mist::opening( img, 1.0 );
25   mist::write_bmp( img, "opening.bmp" );
26
27   size_t label_num;
28   mist::array2< size_t > label;
29   label_num = mist::labeling4( img, label, 100 );
30
31   mist::uniform::random rnd( std::clock( ) );
32   mist::array< mist::rgb< unsigned char > > color_tab( 100 );
33   for( size_t i = 1 ; i < color_tab.size( ) ; i ++ )
34   {
35       color_tab[ i ].r = static_cast< unsigned char >( rnd( 256 ) );
36       color_tab[ i ].g = static_cast< unsigned char >( rnd( 256 ) );
37       color_tab[ i ].b = static_cast< unsigned char >( rnd( 256 ) );
38   }
39
40   mist::array2< mist::rgb< unsigned char > > out( img.width( ),
     img.height( ) );
41   for( size_t i = 0 ; i < label.size( ) ; i ++ )
42   {
43         out[ i ] = color_tab[ label[ i ] % color_tab.size( ) ];
44   }
45   mist::write_bmp( out, "labeling.bmp" );
46
47   std::cout << label_num << std::endl;
48
49   return 0;
50 }
```

サンプルプログラム４

５　行列演算・数値計算

行列演算や数値計算は研究分野を問わず必要となる機会が多い。これらに対して MIST では，行列演算用の matrix コンテナと，数値計算ライブラリ LAPACK（Linear Algebra PACKage）を簡単に利用するためのインタフェースを用意している。ここでは，それらの使用方法を解説する。

MIST のコンテナの１つである matrix を用いることにより，プログラム中での行列演算をシンプルに表現できる。例えば，以下のような行列演算はサンプルプログラム５－１のように書ける。

$$A = \begin{bmatrix} 1 & 4 \\ 2 & 5 \\ 3 & 6 \end{bmatrix},\ B = \begin{bmatrix} 1 & 3 \\ & \\ 2 & 4 \end{bmatrix} \text{のとき} \quad \frac{1}{2}A^{t}A - B \tag{1}$$

matrix コンテナを使用する場合には 01 行が必要である。05 行において 3 行 2 列で各要素の型が“double”の“a”という名前の行列を宣言し，続く 06 から 08 において各要素に値を代入している。09 行ではストリーム演算子“<<”を用いて，標準出力ストリームに“a”を出力している。他の MIST コンテナにもこのようなストリーム演算子が用意されており，コンテナに格納されたデータの標準出力やファイル出力が簡単にできる。行列“b”に関しても“a”と同じように宣言，要素の代入を行っている。最後に 16 行において計算結果を出力しているが，ここでの行列演算の記述は式（1）の記述と同じぐらいシンプルである。

```
01 #include <mist/matrix.h>
02
03 int main( void )
04 {
05   mist::matrix< double > a( 3, 2 );
06   a( 0, 0 ) = 1.0;  a( 0, 1 ) = 4.0;
07   a( 1, 0 ) = 2.0;  a( 1, 1 ) = 5.0;
08   a( 2, 0 ) = 3.0;  a( 2, 1 ) = 6.0;
09   std::cout << a << std::endl << std::endl;
10
11   mist::matrix< double > b( 2, 2 );
12   b( 0, 0 ) = 1.0;  b( 0, 1 ) = 3.0;
13   b( 1, 0 ) = 2.0;  b( 1, 1 ) = 4.0;
14   std::cout << b << std::endl << std::endl;
15
16   std::cout << 1.0 / 2.0 * a.t( ) * a - b << std::endl;
17
18   return 0;
19 }
```

サンプルプログラム５－１

MIST では数値計算に関して，数値計算ライブラリ LAPACK（Linear Algebra PACKage）の扱いやすいインタフェースを提供している。LAPACK は netlib で公開されている歴史と利用実績のある信頼性の高い数値計算ライブラリである。MIST で LAPACK を用いる場合には以下の設定を行う必要がある。

VS.NET2003（Windows）の環境設定

(1) MISTWebサイトのトップページ→左側のダウンロードへ移動→下方ビルド済みLAPACKライブラリLapack-VC.zipをダウンロード→ファイルを適当な場所（この解説では“hogehoge/”とする）に展開。

(2) 上段メニューからプロジェクト→“hoge”（プロジェクト名）のプロパティ→リンカ→全般→追加のライブラリディレクトリに展開したディレクトリパス”hogehoge/Lapack-VC”を追加。

(3) 上段メニューからプロジェクト→“hoge”のプロパティ→リンカ→入力→追加の依存ファイルに“blas.lib lapack.lib libF77.lib libI77.lib”を追加。

(4) サンプルプログラム5－2が正常にビルドできたら終了。

gcc3.0以上（Linux）の環境設定

(1) MISTWebサイトのトップページ→左側のダウンロードへ移動→下方ビルド済みLAPACKライブラリ“Lapack-gcc-LINUX.tar.bz2”をダウンロード→ファイルを適当な場所（この解説では“hogehoge/”とする）に展開。

(2) サンプルプログラム5－2を“hoge.cpp”に記述し，以下のようにコンパイルして，コンパイルエラーが出なければ環境設定は正常に行われている。

```
% g++ hoge.cpp -Wall -DNDEBUG -O2 -Ihogehoge/mist-v1.3.3 -Lhogehoge/Lapack-gcc-LINUX -lm -llapack -lblas -lF77 -lI77 -lpthread
```

以下のような3元1次連立方程式を考える。この方程式はAの逆行列をBの左側から掛けることにより解を得ることができる。サンプルプログラム5－2はこの連立方程式を逆行列による解法と連立方程式を解く関数を用いる方法の2とおりの方法で解いている。

$$\begin{cases} 2x_1+x_2+3x_3=13 \\ -2x_1+5x_2+x_3=11 \\ 3x_1+2x_2-2x_3=1 \end{cases} \Leftrightarrow \quad A=\begin{bmatrix} 2 & 1 & 3 \\ -2 & 5 & 1 \\ 3 & 2 & -2 \end{bmatrix},\ X=\begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix},\ B=\begin{bmatrix} 13 \\ 11 \\ 1 \end{bmatrix} \quad \text{のとき} \quad AX=B \quad (2)$$

MISTの数値計算機能を用いる場合には，02行のヘッダファイルをインクルードする必要がある。16行では逆行列を求める関数“inverse”を用いて方程式を解いている。一方，18行では連立方程式を解く関数“solve”を用いている。ここではごく簡単な例で数値計算機能を紹介したが，その他にもⅣ節2に示したようなさまざまな機能が利用可能である。

```
01 #include <mist/matrix.h>
02 #include <mist/numeric.h>
03
04 int main( void )
```

```
05 {
06   mist::matrix< double > a( 3, 3 );
07   a( 0, 0 ) =  2.0;  a( 0, 1 ) =  1.0;  a( 0, 2 ) =  3.0;
08   a( 1, 0 ) = -2.0;  a( 1, 1 ) =  5.0;  a( 1, 2 ) =  1.0;
09   a( 2, 0 ) =  3.0;  a( 2, 1 ) =  2.0;  a( 2, 2 ) = -2.0;
10
11   mist::matrix< double > b( 3, 1 );
12   b( 0, 0 ) = 13.0;
13   b( 1, 0 ) = 11.0;
14   b( 2, 0 ) =  1.0;
15
16   std::cout << mist::inverse( a ) * b << std::endl << std::endl;
17
18   std::cout << mist::solve( a, b ) << std::endl;
19
20   return 0;
21 }
```

サンプルプログラム５－２

Ⅵ．MIST のこれから

MIST プロジェクトでは，今後もさらなるライブラリの質的向上・利用促進を目指して活動を続けていきたいと思っている。その際に利用者の方々からのフィードバックはたいへん重要である。バグ報告以外でも，MIST を利用してみて不便を感じるところや不足している機能などがあれば，Web サイトへの書き込み，あるいは直接メールでも良いので報告していただきたい。

MIST は音声・画像処理のためのライブラリであるが，現状ではその機能の大部分が画像処理に関連するもので占められている。これは開発メンバの多くが画像処理系の研究室に所属しているためである。今後，MIST がさらなる成長を遂げるためには音声処理機能の強化が必要であると思う。音声処理系の研究室に所属している，あるいは音声処理に興味のある人は，是非 MIST プロジェクトに参加して，音声処理機能の強化へ協力していただきたい。メールでのご連絡をお待ちしている。それ以外にも，ライブラリ開発やプロジェクトの運営に興味があり参加してみたいという方からのご連絡をお待ちしている。

（たかはし　ともかず：名古屋大学大学院情報科学研究科）
（ttakahashi@murase.m.is.nagoya-u.ac.jp）