チュートリアル

ベクトルや行列に関する演算、連立一次方程式や固有値問題などの解をコンピュータ上で高速に得ることは、多くの数値計算の分野で大変重要なテーマとなっています。近年では、これらの線形代数演算をコンピュータのアーキテクチャに特化し、より高速に行うことのできるライブラリが開発されてきています。今回は線形代数演算ライブラリ BLAS, LAPACK に関するチュートリアル（全2回シリーズ）の第1回目として、（独）理化学研究所の中田真秀氏に解説していただきます。

BLAS, LAPACK チュートリアル パート1 （簡単な使い方とプログラミング）

中田 真秀

1　読者の想定

これは、線形代数演算ライブラリ BLAS[1], LAPACK[2] のチュートリアルである。読者の想定は、主に、学生または研究者で、今後このライブラリを活用してみたい人、プログラムを読み書きし始めた人、ブラックボックスとして使っているが中身を知りたい人たち、とした。

2　線形代数の重要性について

線形代数は、現代の科学技術になくてはならないものであり、具体的には、行列、ベクトルなどの演算、連立一次方程式、固有値問題、特異値分解などをコンピュータ上で頻繁に行う。そして、多くの人が知らず知らずのうちにその恩恵を受けている。例を挙げよう。インターネットの最大手サーチエンジン Google の Page Rank ではウェブページに関する相関を行列の固有値を求めてランク付けしているし、3D ゲームで、自分の視点が変わったら景色がどう見えるか、というのは座標変換で、これは行列で表される。CPU、携帯電話、家電などのデバイスの基礎理論は量子力学に支えられているが、そこに必要なのは、大抵、エルミート行列のユニタリ行列による対角化で、巨大な行列の固有値や固有ベクトルを求めることに帰着する。さらに、非線形な問題も、ニュートン法をつかって、連立一次方程式を繰り返し解くことで解いたりする。つまり線形代数は様々な分野で様々な形で現れる重要なものなのである。

3　BLAS, LAPACK とは何か：世界最高の線形代数演算パッケージ

コンピュータで線形代数演算を行うライブラリは BLAS[1] および LAPACK[2] が、間違いなく世界最高のものである。無料で入手でき、品質、信頼性が高く、網羅している演算の種類も多く、サードパーティによる高速な実装もあるからだ。プログラムは一度は自分で書いた方がいいが、このクオリティは並大抵のものではないので、中身を勉強しつつこのライブラリに頼ることは良いことである。ただ、BLAS, LAPACK は密行列向けであり、疎行列はサポートされていない点に注意。以下に簡単に解説する。

(1) BLAS とは？

まず BLAS（Basic Linear Algebra Subprograms）だが、その名のとおり基礎的な線形代数のサブプログラムであり、FORTRAN77 で様々なルーチンの仕様を提供している。つまり、あるルーチンはこの動きをする、という定義をしている。また、サブプログラムとあるよう、BLAS には、行列やベクトルの定数倍、和、積など非常に簡単なルーチンのみが用意されている。目的はビルディングブロックとして、より高度なプログラムをユーザーに作ってもらうことにある。さらに、BLAS は手本や見本となるような、参照実装も提供されており（reference BLAS と呼ばれる）、誰でも入手、動作確認ができる。参照実装はとても美しく書かれており、プログラム自身がそのままドキュメントとできるくらいである。参照実装であることと、その美しさについては頻繁に見落とされるが、これは本質的なことである。

さて、BLAS には Level 1, Level 2 そして Level 3 と三種類のものがある。x, y はベクトル α, β は定数、A, B, C

筆者紹介

なかた まほ
理化学研究所 情報基盤センター 協力研究員。博士（工学）。専門は量子化学（縮約密度行列の変分法）、最適化（半正定値計画）、ハイパフォーマンスコンピューティング（線形代数）。BLAS、LAPACK の高精度版の構築と応用に興味を持っている。
http://accc.riken.jp/maho/

は行列とする。

まず、Level 1 BLASは、ベクトル-ベクトル演算のルーチン群である、ベクトルの加算（DAXPY）、

$$y \leftarrow \alpha x + y \tag{1}$$

や内積計算（DDOT）

$$dot \leftarrow x^T y \tag{2}$$

など15種類あり、さらに単精度、倍精度、複素数単精度、複素数倍精度についての4通りの組み合わせがある。

次にLevel 2 BLASであるが、行列-ベクトル演算のルーチン群であり、行列-ベクトル積（DGEMV）

$$y \leftarrow \alpha Ax + \beta y \tag{3}$$

や、上三角行列の連立一次方程式を解く（DTRSV）

$$x \leftarrow A^{-1} b \tag{4}$$

など25種類あり、同じように4通りの組み合わせがある。最後にLevel 3 BLASは行列-行列演算のルーチン群であり、行列-行列積（DGEMM）、

$$C \leftarrow \alpha AB + \beta C \tag{5}$$

やDSYRK、

$$C \leftarrow \alpha AA^T + \beta C \tag{6}$$

上三角行列の連立一次方程式を解くDTRSM

$$B \leftarrow \alpha A^{-1} B \tag{7}$$

など、9種類、さらに加えて精度などによる組み合わせがある。どんなルーチンがあるかを参照したい場合はQuick referenceがあるのでそれを参照としていただきたい。

(2) LAPACKとは？

LAPACK（Linear Algebra PACKage）もその名の通り、線形代数パッケージである。BLASをビルディングブロックとして使いつつ、より高度な問題である連立一次方程式、最小二乗法、固有値問題、特異値問題を解くことができる。また、それに伴った行列の分解（LU分解、コレスキー分解、QR分解、特異値分解、Schur分解、一般化Schur分解）、さらには条件数の推定ルーチン、逆行列計算など、様々な下請けルーチン群も提供する。品質保証も非常に精密かつ系統的で、信頼のおけるルーチン群である。また、BLASとは違い、バージョン3.2からはFortran 90で書かれ、2011/2/15現在バージョン3.3.0が出ている。特に今回は標準的なCのAPI, CS分解、Level 3 BLASを多用したソルバ、完全なスレッドセーフの実現、がハイライトとされている。3.3.0は、1632ものルーチンからなっている。単精度、倍精度、複素数単精度、複素数倍精度について各400個程度のルーチンがある。パソコンからスーパーコンピュータまで、様々なCPU, OS上で動き、webサイトはなんと9000万ヒットである。世界中の人に愛されている素晴らしいライブラリである。

4 BLAS, LAPACKを使う上での注意点

BLAS, LAPACKを扱う上でいくつか注意すべき点を挙げる。FORTRANの細かい言語仕様が、意外と落とし穴になる。

(1) Column major, row majorに注意

行列は2次元だが、コンピュータのメモリは1次元的である。次のような行列を

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{pmatrix}$$

考えるとき、どのようにメモリに格納するかの違いがcolumn major, row majorである。アドレスの小さい順から

1, 4, 2, 5, 3, 6

のように格納されるのが、図1にもあるよう、column majorであり、アドレスの小さい順から

1, 2, 3, 4, 5, 6

のように格納されるのが、図2にもあるよう、row majorである。FROTRAN, Matlab, Octaveではcolumn majorとなっており、C, C++では普通row majorであるので、C, C++から呼び出すときや、高速化のため、連続メモリアクセスを行いたいときなどに意識する必要が出てくる。

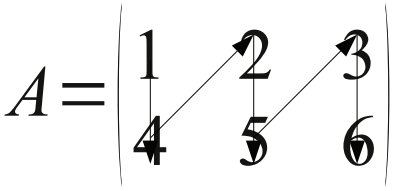

図1 Column Major：行列のデータは列方向にメモリに格納される。

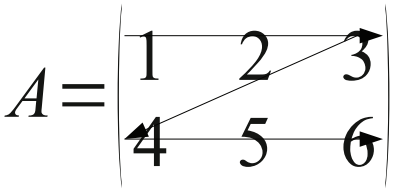

図2 Row Major：行列のデータは行方向にメモリに格納される。

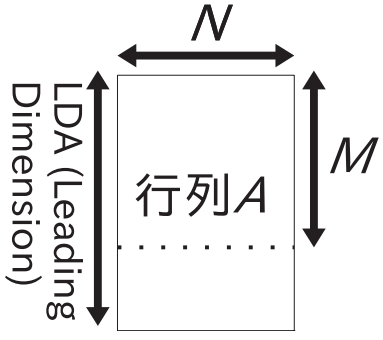

図3 Leading Dimensionの考え方：$M \times N$行列Aは、より大きな$LDA \times N$行列の部分行列と扱うことがある。

(2) 行列のLeading dimensionとは？

実際の計算で、行列をより大きな部分行列として考えると便利なことがある。そのために、“leading dimension”が設定されている（図3）。BLAS, LAPACKなどで現れる、LDA, LDBなどの引数がそれである。図の例では、$M \times N$の行列Aは$LDA \times N$の行列の部分行列となっている。行列$A(i, j)$をC/C++からアクセスする際は、$A[i+j*m]$ではなくて、

$$A[i+j*lda]$$

とアクセスしなければならない。このような手法はLU分解、Chokesky分解など、LAPACKでは多用されている。

(3) FORTRAN, C, C++ の配列のスタートの違い

FORTRANでは配列は1からスタートするが、C, C++では、0からスタートする。例えば

- ループの書き方が一般的には1からNまで（FORTRAN）か、0からn未満か（C, C++）。
- ベクトルのx_i要素へのアクセスはFORTRANでは$X(I)$だが、C, C++では$x[i\text{-}1]$となる。
- 行列の$A_{i,j}$要素へのアクセスはFORTRANではA(I,J)だが、C, C++ではcolumn majorとして$A[i-1+(j-1)*lda]$とする。

などである。

5 BLAS, LAPACKの現状について

- 高速なBLAS, LAPACKについて：reference BLASは仕様書そのままなので、非常に低速である。なぜならば最近のコンピュータの特徴は、マルチコア、深いパイプライン、メモリのスピードによる階層構造；レジスタ、L1, L2, L3キャッシュなどを考えに入れていないからである（BLAS自体は1970年代から開発されているから、当然だが）。これらについてIntel社のMKL（Math Kernel Library）、AMD社のACML（AMD Core Math Library）、GotoBLAS2やATLASなどがある。スパコンなどにはベンダーが大抵最適化したBLAS, LAPACKを用意してあるのでそれを使うと良い。おすすめはGotoBLAS2またはIntel MKLである。
- GotoBLAS2[3]：GotoBLAS2は後藤和茂氏による、BLAS, LAPACKのおそらく世界で一番高速な実装である。様々なCPU, OSに対応している。さらにフリーソフトウェア（オープンソース）でもある。“BLAS”とあるが、LAPACK3.1.1も一部のルーチンを加速してあり使い勝手もよい。ただし開発は中止されたので最新のプロセッサには対応していないこともある。
- ATLAS[4]：R.Clint Whaley氏による、オートチューニング機構で高速化したBLAS。パラメータを多く用意し、それについて実機で実行させて時間を測定し、一番速いパラメータを取る、という風に作成する。LAPACKの一部のルーチンも高速化されている。多くのコンピュータのBLAS環境を劇的に改善した、パイオニア的存在。2001年からフリーソフトウェアだったので多くのコンピュータに入っている。残念ながら、ビルドに時間がかかったり、各マシンで個別に最適化しなければならないので不便なのと、チューニングしたGotoBLAS2よりは数%から数10%低速であるという欠点もある。
- BLAS, LAPACKを利用したソフトウェア：著名な計算プログラムパッケージは大抵BLAS, LAPACKを利用している。例えば物理、化学ではGaussian, Gamess, ADF, VASPなど線形計画問題のCPLEX, NUOPT, GLPKなど。またRuby, Python, Perl, Java, C, Mathematica, Maple, Matlab, R, octave, SciLabなど、さまざま高級言語からも使われている。
- Top 500[5]：世界で一番高速なコンピューターを決めるTop 500では、LINPACKのパフォーマンスを測定してランキングが定まる。ここで一番重要なのは、DGEMMと呼ばれる行列-行列積のパフォーマンスで、このチューンナップが重要である。政治的にも重要のようである。
- 並列版BLAS, LAPACK[6]：ScaLAPACKというプロジェクトがある。これを用いるとより巨大な行列の演算が行える。
- GPU向けBLAS, LAPACK[7]：近年CPUの性能が頭打ちになっている一方、グラフィックスボードの高性能化が著しくそれを計算に使うことも行われている。CPUに比べ、数倍～10倍程度高速かつ安価なので、大変注目されている。nVidia社のGPUを用いたMAGMAプロジェクトが有望だと考えている。
- 高精度BLAS, LAPACK[8]：エクサ級のコンピュータを一週間走らせると10^{23}回の演算を行う。倍精度は16桁なので、計算誤差の蓄積が懸念される。それ以前にもそもそもKrylov部分空間法や、半正定値計画法など、誤差の溜まりやすい問題もあり、高精度計算の重要性が増している。これに対してBLAS,LAPACKを高精度に拡張したMPACKを筆者が開発している。

6 Ubuntu 10.04x86（Lucid Lynx）デスクトップ版でBLAS, LAPACKを実際に使ってみる

ここではBLAS, LAPACKを実際に使ってみることにする。まず前準備の後、GotoBLAS2のインストール、次にその威力を対話的な高級言語Octave[9]から体験する。そしてC++から行列積、対角化を行う。残念ながら、BLAS, LAPACK環境はOSなどに強く依存するので、MacとLinuxとWindowsではやり方が大きく違う。今回はなるべく多くの人が手軽に試せるようにソフトはすべて高品質かつ無料で入手も容易なものを示した。

(1) 前準備

OSであるが、Ubuntu 10.04 x86（Lucid Lynx）デスクトップ版[10]というOSをインストールしたマシンを用

意していただきたい。いつも使っている環境を壊さないためには、仮想マシンのVirtualBox[11]などで試すのがよいだろう。ただし仮想環境ではどうしてもパフォーマンスは落ちることに注意。ホームディレクトリは、

```
/home/maho
```

となっている。適宜、都合で読み替えていただきたい。さて、いくつか標準で入っていないソフトのインストールをする。メニューから[アプリケーション]→[アクセサリ]→[端末]、で端末を開く。そこで作業をする。“$”はコマンドプロントなので、入力しない。“\”は紙面の都合上の折り返しである。さて、以下のコマンドを入力する。

```
$ sudo apt-get install patch gfortran g++ \
 libblas-dev octave3.2
```

(2) GotoBLAS2のインストール

GotoBLAS2は執筆現在1.13が最新バージョンで

```
http://www.tacc.utexas.edu/tacc-projects/gotoblas2/
```

からダウンロードした後、端末から以下のように入力する。

```
$ cd ; cp <somewhere>/GotoBLAS2-1.13_bsd.tar.gz.
$ tar xvfz GotoBLAS2-1.13_bsd.tar.gz
$ cd GotoBLAS2
$ ./quickbuild.64bit
...
ln -fs libgoto2_nehalemp-r1.13.so libgoto2.so

 GotoBLAS build complete.

  OS               ...Linux
  Architecture     ...x86_64
  BINARY           ...64bit
  C compiler       ...GCC(command line : gcc)
  Fortran compiler...GFORTRAN \
            (command line : gfortran)
  Library Name     ...libgoto2_nehalemp-r1.13.a\
            (Multi threaded; Max num-threads is 8)
```

環境によって若干異なるが、大体上のようになっていれば、ビルドは成功である。Intel Core i7 920（2.66GHz）で、所要時間は2分程度であった。

(3) OctaveでGotoBLAS2の威力を体感する。

Octave[9]は対話的な数値計算プログラムで、手軽にベクトル、行列演算などができる。Matlab互換の高級言語で、内部からBLAS, LAPACKを用いている。今回、マシンはIntel Core i7 920（2.66GHz, TurboBoostオフ時理論性能値42.56GFlops, 1GFlopsとは1秒間に十億回演算ができることを表す。回数が多い方が性能は高い）を利用した。これでreference BLAS, ATLAS, GotoBLAS2の違いを比較する。入れ替えただけで、これほど差が出るというのを体感していただきたい。

まず、ランダム行列を二つ生成し、その積を計算し、何GFlopsでるか見よう。BLASのDGEMMのパフォーマンスを見ることになる。BLAS, LAPACKライブラリの切り替えは、referenceの場合、

```
$ LD_PRELOAD=/usr/lib/libblas.so:\
/usr/lib/liblapack.so; export LD_PRELOAD
```

で、ATLAS（Ubuntuデフォルト）

```
$ LD_PRELOAD=/usr/lib/atlas/libblas.so:/usr/\
lib/atlas/liblapack.so; export LD_PRELOAD
```

や、

```
$ LD_PRELOAD=/home/maho/GotoBLAS2/\
libgoto2.so; export LD_PRELOAD
```

のように行う。まずは、一番遅いレファリンスBLASの場合。

```
$ octave
...途中略...
octave:1> n=4000; A=rand(n); B=rand(n);
octave:2> tic(); C=A*B; t=toc(); GFLOPS=2*n^3/t*1e-9
GFLOPS = 1.6865
```

理論性能値のたった4%程度しかでていない。Ubuntu付属のATLASではもう少し高速になるが、

```
$ LD_PRELOAD=/usr/lib/atlas/libblas.so; \
export LD_PRELOAD
$ octave
...途中略...
octave:1> n=4000; A=rand(n); B=rand(n);
octave:2> tic(); C=A*B; t=toc(); GFLOPS=2*n^3/t*1e-9
GFLOPS = 7.0291
```

理論性能値の16.5%程度しか出ていない。ATLASは使うマシン上でビルドしないと性能が出ないので、この比較はフェアでない。詳細は省くが、自分でATLAS 3.9.36をビルドし直した場合

```
octave:1> n=4000; A=rand(n); B=rand(n);
octave:2> tic(); C=A*B; t=toc(); GFLOPS=2*n^3/t*1e-9
GFLOPS = 32.824
```

となり、理論性能値と比較して77.1%であった。だが、GotoBLAS2を用いると、パフォーマンスがさらに上がる。

```
$ LD_PRELOAD=/home/maho/GotoBLAS2/libgoto2.so ;\
export LD_PRELOAD
$ octave
...途中略...
octave:1> n=4000; A=rand(n); B=rand(n);
octave:2> tic(); C=A*B; t=toc(); GFLOPS=2*n^3/t*1e-9
GFLOPS =  39.068
```

理論性能値の91.2%と驚異的なパフォーマンスを示した。

次に、対称行列の固有値を求める時間を比較しよう。

```
octave:1> n=4000; A=rand(n); B=(A+A')/2;
octave:2> tic(); eig(B); toc();
```

GotoBLAS2では

```
Elapsed time is 8.05604 seconds.
```

reference BLASでは、

```
Elapsed time is 37.7076 seconds.
```

ATLASでは

```
Elapsed time is 33.5624 seconds.
```

自分でビルドしたATLAS 3.9.36では、

```
Elapsed time is 26.4307 seconds.
```

であった。GotoBLAS2を使った方がデフォルトのATLASと比べて4.2倍高速で、ATLASはreference BLASとそんなに変化が無かった。GotoBLAS2の圧倒的なパフォーマンスについてわかっていただけたと思う。

(4) BLAS実習：C++から行列-行列積DGEMMを使う

行列-行列積DGEMMを使ってみよう。ここでは、

$$A=\begin{pmatrix}1 & 8 & 3\\ 2 & 10 & 8\\ 9 & -5 & -1\end{pmatrix}$$

$$B=\begin{pmatrix}9 & 8 & 3\\ 3 & 11 & 2.3\\ -8 & 6 & 1\end{pmatrix}$$

$$C=\begin{pmatrix}3 & 3 & 1.2\\ 8 & 4 & 8\\ 6 & 1 & -2\end{pmatrix}$$

α=3, β= −2として、

$$C \leftarrow \alpha AB + \beta C$$

を計算するプログラムを書いてみる。答えは

$$\begin{pmatrix}21 & 336 & 70.8\\ -64 & 514 & 95\\ 210 & 31 & 47.5\end{pmatrix}$$

である。さて、図4のようにエディタなどで入力し、

```
$ g++ -static -pthread dgemm_demo.cpp -o \
dgemm_demo -L/home/maho/GotoBLAS2/ -lgoto2
```

でコンパイルができる。何もメッセージが出ないなら、コンパイルは成功である。実行は以下のようになっていればよい。なお、Octaveに、この結果をそのままコピー&ペーストすれば答えをチェックできる。

```
$ ./dgemm_demo
# dgemm demo...
A =[ [ 1.00e+00, 8.00e+00, 3.00e+00];\
     [ 2.00e+00, 1.00e+01, 8.00e+00];\
     [ 9.00e+00, -5.00e+00, -1.00e+00] ]
B =[ [ 9.00e+00, 8.00e+00, 3.00e+00];\
     [ 3.00e+00, 1.10e+01, 2.30e+00];\
     [ -8.00e+00, 6.00e+00, 1.00e+00] ]
C =[ [ 3.00e+00, 3.00e+00, 1.20e+00];\
     [ 8.00e+00, 4.00e+00, 8.00e+00];\
     [ 6.00e+00, 1.00e+00, -2.00e+00]]
alpha = 3.000e+00
beta  = -2.000e+00
ans=[ [ 2.10e+01, 3.36e+02, 7.08e+01];\
      [ -6.40e+01, 5.14e+02, 9.50e+01];\
      [ 2.10e+02, 3.10e+01, 4.75e+01] ]
#check by Matlab/Octave by:
alpha * A * B + beta * C
```

(5) LAPACK実習：C++から行列の固有ベクトル、固有値を求めるDSYEVを使ってみる

次はLAPACK実習としてC++から行列の対角化DSYEVを使ってみる。

$$A=\begin{pmatrix}1 & 2 & 3\\ 2 & 5 & 4\\ 3 & 4 & 6\end{pmatrix}$$

```
// dgemm test public domain
#include <stdio.h>
extern "C" {
#define ADD_
#include <cblas_f77.h>
}
//Matlab/Octave format
void printmat(int N, int M, double *A, int LDA) {
  double mtmp;
  printf("[ ");
  for (int i = 0; i < N; i++) {
    printf("[ ");
    for (int j = 0; j < M; j++) {
      mtmp = A[i + j * LDA];
      printf("%5.2e", mtmp);
      if (j < M - 1) printf(", ");
    } if (i < N - 1) printf("]; ");
    else printf("] ");
  } printf("]");
}
int main()
{
  int n = 3; double alpha, beta;
  double *A = new double[n*n];
  double *B = new double[n*n];
  double *C = new double[n*n];

  A[0+0*n]=1; A[0+1*n]= 8; A[0+2*n]= 3;
  A[1+0*n]=2; A[1+1*n]=10; A[1+2*n]= 8;
  A[2+0*n]=9; A[2+1*n]=-5; A[2+2*n]=-1;

  B[0+0*n]= 9; B[0+1*n]= 8; B[0+2*n]=3;
  B[1+0*n]= 3; B[1+1*n]=11; B[1+2*n]=2.3;
  B[2+0*n]=-8; B[2+1*n]= 6; B[2+2*n]=1;

  C[0+0*n]=3; C[0+1*n]=3; C[0+2*n]=1.2;
  C[1+0*n]=8; C[1+1*n]=4; C[1+2*n]=8;
  C[2+0*n]=6; C[2+1*n]=1; C[2+2*n]=-2;

  printf("# dgemm demo...\n");
  printf("A =");printmat(n,n,A,n);printf("\n");
  printf("B =");printmat(n,n,B,n);printf("\n");
  printf("C =");printmat(n,n,C,n);printf("\n");
  alpha = 3.0; beta = -2.0;
  F77_dgemm("n", "n", &n, &n, &n, &alpha,
        A, &n, B, &n, &beta, C, &n);
  printf("alpha = %5.3e\n", alpha);
  printf("beta  = %5.3e\n", beta);
  printf("ans="); printmat(n,n,C,n);
  printf("\n");
```

```
  printf("#check by Matlab/Octave by:\n");
  printf("alpha * A * B + beta * C =\n");
  delete[]C; delete[]B; delete[]A;
}
```

図4 C++でのDGEMMのサンプル。行列-行列積を求める。ファイル名は“*dgemm_demo.cpp*”とすること。

の固有ベクトル、固有値を求めてみる。固有値は、

$-0.40973, 1.57715, 10.83258$

で、固有ベクトルv_1, v_2, v_3は

$v_1 = (-0.914357, 0.216411, 0.342225)$
$v_2 = (0.040122, -0.792606, 0.608413)$
$v_3 = (0.402916, 0.570037, 0.716042)$

である。図5のようにエディタなどで入力し、

```
$ g++ -static -pthread eigenvalue_demo.cpp \
-o eigenvalue_demo -L/home/maho/GotoBLAS2/ \
-lgoto2 -lgfortran
```

でコンパイルができる。何もメッセージが出ないなら、コンパイルは成功である。実行は以下のようになっていればよい。同じようにOctaveに、この結果をそのままコピー&ペーストすれば答えをチェックできる。

```
$ ./eigenvalue_demo
A =[ [ 1.00e+00, 2.00e+00, 3.00e+00];\
     [ 2.00e+00, 5.00e+00, 4.00e+00];\
     [ 3.00e+00, 4.00e+00, 6.00e+00] ]
#eigenvalues
w =[ [ -4.10e-01]; [ 1.58e+00]; [ 1.08e+01] ]
#eigenvecs
U =[ [ -9.14e-01, 2.16e-01, 3.42e-01];\
     [ 4.01e-02, -7.93e-01, 6.08e-01];\
     [ 4.03e-01, 5.70e-01, 7.16e-01] ]
#Check Matlab/Octave by:
eig(A)
U' *A*U
```

```
//dsyev test public domain
#include <iostream>
#include <stdio.h>
extern "C" int dsyev_(const char *jobz, const char
*uplo, int *n, double *a, int *lda, double *w,
double *work, int *lwork, int *info);
//Matlab/Octave format
void printmat(int N, int M, double *A, int LDA) {
  double mtmp;
  printf("[ ");
  for (int i = 0; i < N;  i++) {
    printf("[ ");
    for (int j = 0; j < M; j++) {
      mtmp = A[i + j * LDA];
      printf("%5.2e", mtmp);
      if (j < M - 1) printf(", ");
    } if (i < N - 1) printf("]; ");
    else printf("] ");
  } printf("]");
}
int main()
{
  int n = 3;
  int lwork, info;
  double *A = new double[n*n];
  double *w = new double[n];
//setting A matrix
  A[0+0*n]=1;A[0+1*n]=2;A[0+2*n]=3;
  A[1+0*n]=2;A[1+1*n]=5;A[1+2*n]=4;
  A[2+0*n]=3;A[2+1*n]=4;A[2+2*n]=6;

  printf("A ="); printmat(n, n, A, n);
  printf("\n");
  lwork = -1;
  double *work = new double[1];
  dsyev_("V", "U", &n, A, &n, w, work, &lwork, &info);
  lwork = (int)work[0];
  delete[]work;
  work = new double[std::max((int) 1, lwork)];
//get Eigenvalue
  dsyev_("V", "U", &n, A, &n, w, work, &lwork, &info);
//print out some results.
  printf("#eigenvalues \n"); printf("w =");
  printmat(n, 1, w, 1); printf("\n");
  printf("#eigenvecs \n"); printf("U =");
  printmat(n, n, A, n); printf("\n");
  printf("#Check Matlab/Octave by:\n");
  printf("eig(A)\n");
  printf("U'*A*U\n");
  delete[]work;
  delete[]w;
  delete[]A;
}
```

図5 C++でのDSYEV対角化、固有ベクトルを求めるサンプル。ファイル名は“*eigenvalue_demo.cpp*”とすること。

7 終わりと次回予告

皆様の参考になることを願いつつ、駆け足でBLAS、LAPACKについて紹介した。

もし、ご意見、ご希望、間違い、今後取り上げてほしい話題などがあれば気軽にメールを送って頂ければ幸いである。

謝辞

日頃から多くの指導、助言、励ましを頂いた、藤澤克樹先生、後藤和茂氏、姫野龍太郎先生、高雄保嘉氏、安井雄一郎君に深く感謝する。

■参考文献

[1] http://www.netlib.org/blas/
[2] http://www.netlib.org/lapack/
[3] http://www.tacc.utexas.edu/tacc-projects/gotoblas2/
[4] http://math-atlas.sourceforge.net/
[5] http://www.top500.org/
[6] http://www.netlib.org/scalapack/
[7] http://icl.cs.utk.edu/magma/
[8] http://mplapack.sourceforge.net/
[9] http://www.gnu.org/software/octave/
[10] http://www.ubuntulinux.jp/
[11] http://www.virtualbox.org/