線形代数演算ライブラリ BLAS と LAPACK の基礎と実践 1

中田真秀

理化学研究所, 情報基盤センター

2015/05/21 CMSI 教育計算科学技術特論 A

RIKEN

BLAS LAPACK 入門 (I) 講義内容

- 線形代数の歴史と重要性
- コンピュータでの数値計算と、線形代数演算
- BLAS, LAPACK の紹介。
- BLAS を使ってみる:行列-行列積
- LAPACK を使ってみる:対称行列の対角化
- まとめと次回予告

線形代数の歴史と重要性

- 人類は、線形代数を有志以来やってきた。エジプトが最古 (パピルス)、中国もガウスの消去法は 1000 年以上前に知っていた (九章算術; 紀元前 1 世紀から紀元後 2 世紀ころ)。
- あらゆる分野に線形代数がでてくる:物理、化学、工学、生物学、経済、経営...
- コンピュータが生まれ、線形代数演算をコンピュータ上で高速に、大量にやらせることが重要になった。
- 主にスピードおよび規模を追求してきた。これがコンピュータの歴史。

線形代数の歴史と重要性

- 人類は、線形代数を有志以来やってきた。エジプトが最古 (パピルス)、中国もガウスの消去法は 1000 年以上前に知っていた (九章算術; 紀元前 1 世紀から紀元後 2 世紀ころ)。
- あらゆる分野に線形代数がでてくる:物理、化学、工学、生物学、経済、経営...
- コンピュータが生まれ、線形代数演算をコンピュータ上で高速に、大量にやらせることが重要になった。
- 主にスピードおよび規模を追求してきた。これがコンピュータの歴史。

線形代数の歴史と重要性

- 人類は、線形代数を有志以来やってきた。エジプトが最古 (パピルス)、中国もガウスの消去法は 1000 年以上前に知っていた (九章算術; 紀元前 1 世紀から紀元後 2 世紀ころ)。
- あらゆる分野に線形代数がでてくる:物理、化学、工学、生物学、経済、経営...
- コンピュータが生まれ、線形代数演算をコンピュータ上で高速に、大量にやらせることが重要になった。
- 主にスピードおよび規模を追求してきた。これがコンピュータの歴史。

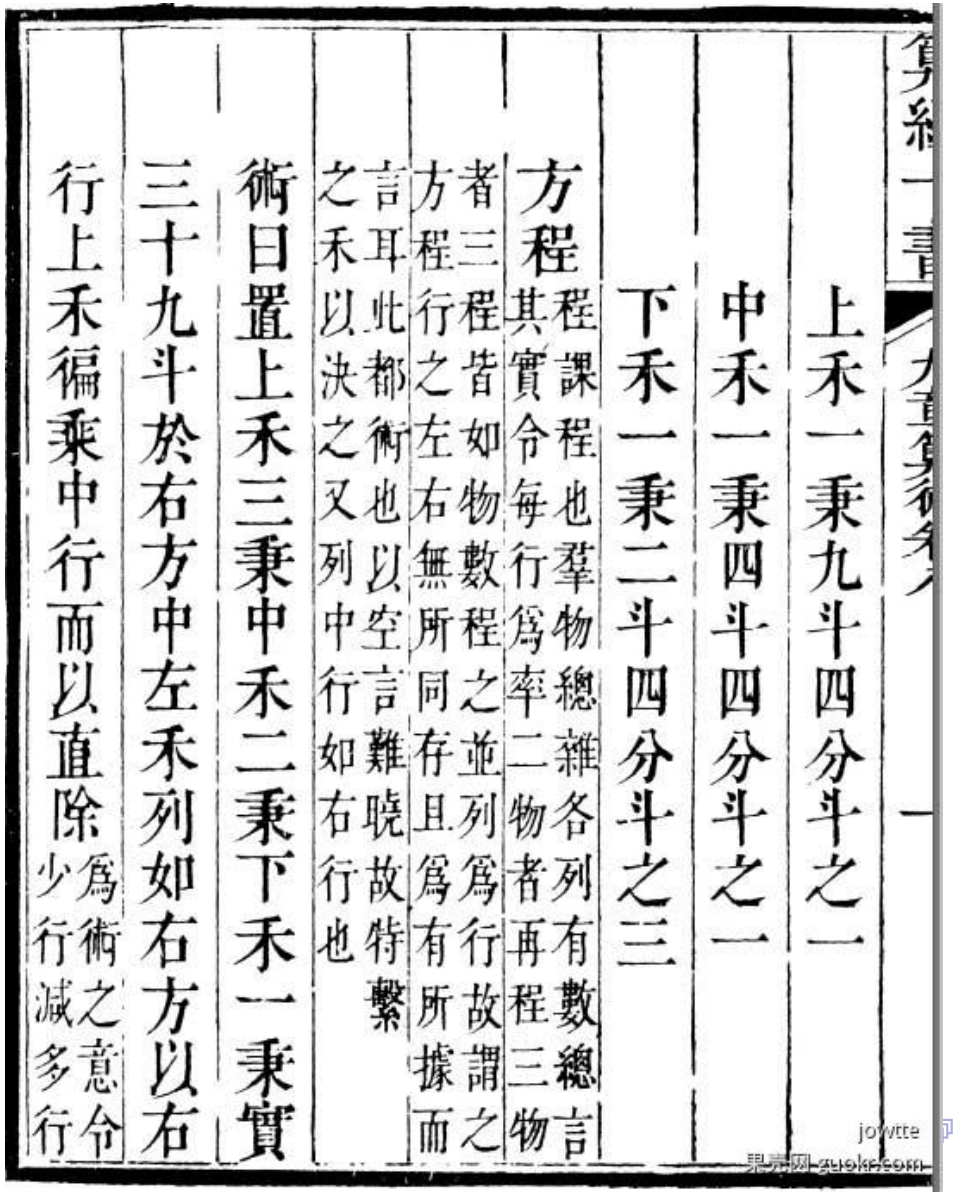
線形代数の歴史と重要性

- 人類は、線形代数を有志以来やってきた。エジプトが最古 (パピルス)、中国もガウスの消去法は 1000 年以上前に知っていた (九章算術; 紀元前 1 世紀から紀元後 2 世紀ころ)。
- あらゆる分野に線形代数がでてくる:物理、化学、工学、生物学、経済、経営...
- コンピュータが生まれ、線形代数演算をコンピュータ上で高速に、大量にやらせることが重要になった。
- 主にスピードおよび規模を追求してきた。これがコンピュータの歴史。

線形代数の歴史と重要性

- 人類は、線形代数を有志以来やってきた。エジプトが最古 (パピルス)、中国もガウスの消去法は 1000 年以上前に知っていた (九章算術; 紀元前 1 世紀から紀元後 2 世紀ころ)。
- あらゆる分野に線形代数がでてくる:物理、化学、工学、生物学、経済、経営...
- コンピュータが生まれ、線形代数演算をコンピュータ上で高速に、大量にやらせることが重要になった。
- 主にスピードおよび規模を追求してきた。これがコンピュータの歴史。

九章算術 (中国、紀元前１世紀から紀元後２世紀ころ)、方程から



九章算術一 方程

上禾一秉九斗四分斗之一
中禾一秉四斗四分斗之一
下禾一秉二斗四分斗之三

方程 程課程也羣物總雜各列有數總言其實令每行爲率二物者再程三物者三程皆如物數程之並列爲行故謂之方程行之左右無所同存且爲有所據而言耳此都術也以空言難曉故特繫之禾以決之又列中行如右行也

術曰置上禾三秉中禾二秉下禾一秉實三十九斗於右方中左禾列如右方以右行上禾徧乘中行而以直除爲術之意令少行減多行

九章算術 (中国、紀元前１世紀から紀元後２世紀ころ)、方程から

- 今有上禾三秉，中禾二秉，下禾一秉，實三十九斗；上禾二秉，中禾三秉，下禾一秉，實三十四斗；上禾一秉，中禾二秉，下禾三秉，實二十六斗。問上、中、下禾實一秉各幾何？
- 答曰:上禾一秉，九斗、四分斗之一，中禾一秉，四斗、四分斗之一，下禾一秉，二斗、四分斗之三。
- 方程術曰，置上禾三秉，中禾二秉，下禾一秉，實三十九斗，於右方。中、左禾列如右方。以右行上禾遍乘中行而以直除。又乘其次，亦以直除。然以中行中禾不盡者遍乘左行而以直除。左方下禾不盡者，上為法，下為實。實即下禾之實。求中禾，以法乘中行下實，而除下禾之實。餘如中禾秉數而一，即中禾之實。求上禾亦以法乘右行下實，而除下禾、中禾之實。餘如上禾秉數而一，即上禾之實。實皆如法，各得一斗。

九章算術 (紀元前 1 世紀から紀元後 2 世紀ころ)、方程、中田+Google trans. 訳

- 問: 3 束の上質のキビ、2 束の中質のキビ、1 束の低質のキビが 39 個のバケツに入っている。2 束の上質のキビ、3 束の中質のキビ、1 束の低質のキビが 34 個のバケツに入っている。1 束の上質のキビ、2 束の中質のキビ、3 束の低質のキビが 26 個のバケツに入っている。上質、中質、低質のキビ一束はそれぞれバケツいくつになるか。
- 答: 上質 $9\frac{1}{4}$, 中質 $4\frac{1}{4}$, 低質 $2\frac{3}{4}$
- 上質のキビ 3 束、中質のキビ 3 束、低質のキビ 1 束を 39 バケツを右行に置く。中行、左行も右のように並べる。右の上質を中行にかけ、右行で引く。また左行にもかけて右行から引く。次に、中行の中質のキビの余りを左行にかけて、中行で引く。左の低質に余りがあるのでそして、割れば求まる (実を法で割る)。以下略

現代風に...

九章算術 (紀元前 1 世紀から紀元後 2 世紀ころ)、方程、中田+Google trans. 訳

- 問:

$$\begin{cases} 3x + 2y + z = 39 \cdots (右) \\ 2x + 3y + z = 34 \cdots (中) \\ x + 2y + 3z = 26 \cdots (左) \end{cases}$$

- (右) はそのまま、(中) は (中) を 3 倍したものから (右) を 2 倍したものを引く、(左) を 3 倍して (左) から (右) を引く。

$$\begin{array}{ll} 3(2x + 3y + z = 34) & 3(x + 2y + 3z = 39) \\ 2(3x + 2y + z = 39) & 3x + 2y + z = 39 \\ \hline 5y + z = 24 \cdots (中) & 4y + 8z = 39 \cdots (左) \end{array}$$

- それから、(左) を 5 倍する。

$$\begin{cases} 3x + 2y + z = 39 \cdots (右) \\ 5y + z = 24 \cdots (中) \\ 20y + 40z = 195 \cdots (左) \end{cases}$$

- (左)-(中)x4 をする

$$36z = 99$$

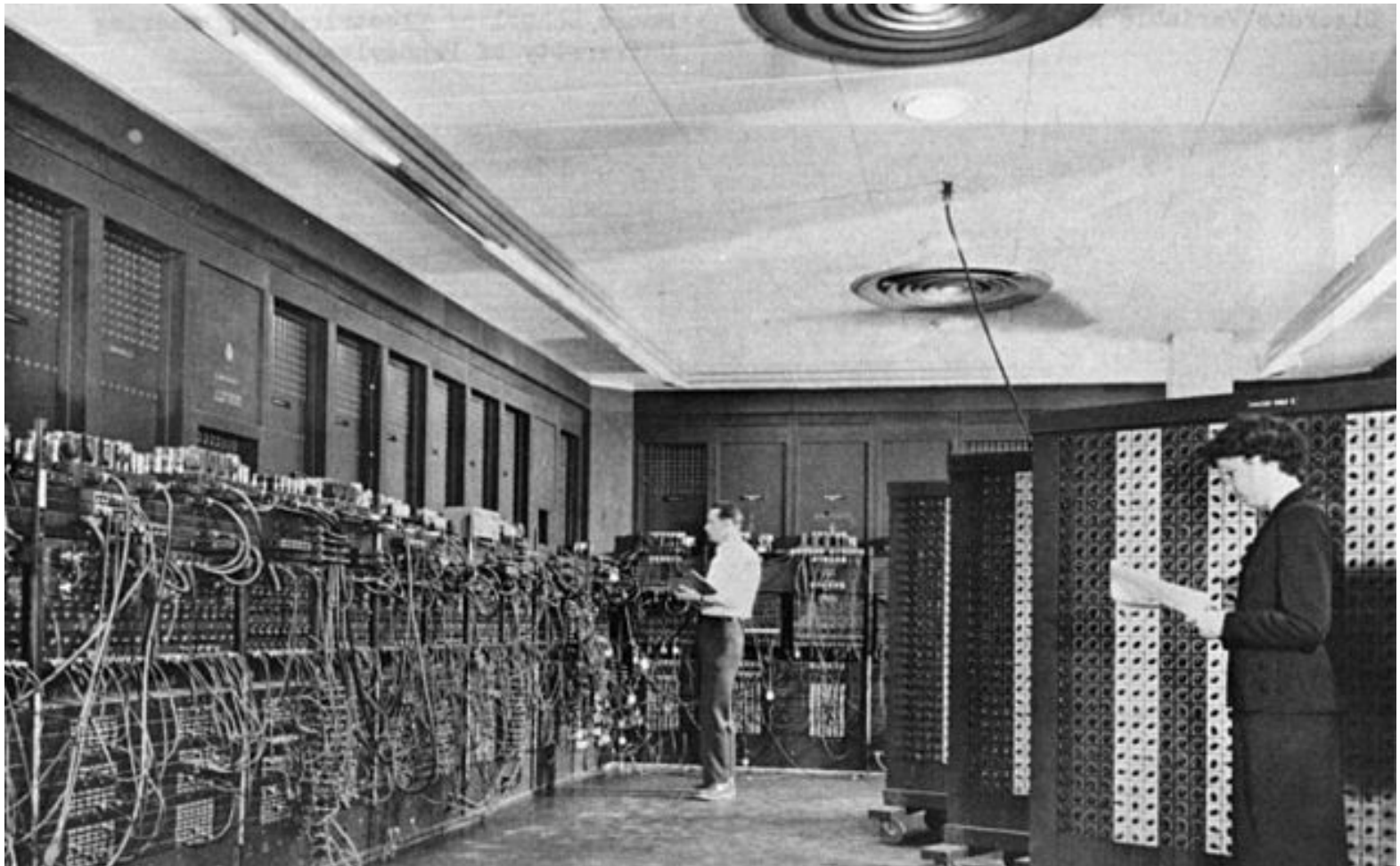
後は略

近現代の線形代数

- 1693 年ライプニッツ、1750 年頃クラメール、1888 年ペアノ
- 1900 年有限のベクトル空間の定義
- 大雑把に:無限次元の線形代数=ヒルベルト空間 (=量子力学)
- 1950 年代〜コンピュータ上での線形代数の発達 (LU 分解、固有値分解など)

ENIAC で計算を行なっている写真

ENIAC(エニアック、Electronic Numerical Integrator and Computer、1946 年) は、アメリカで開発された黎明期の電子計算機（コンピュータ）10 桁の数値同士の乗算は 14 サイクル（2800 マイクロ秒）かかり、毎秒 357 回ということになる (Wikipedia より)。



コンピュータでの実数演算と線形代数演算について

コンピュータでの実数演算はどうするか

コンピュータは有限なので実数の一部しか取り扱うことができないため、近似的な取り扱いをする

- 浮動小数点数: コンピュータ上での実数の近似表現の一つ。

$$1.01010101 \times 2^3$$

- とても実用的で、よく使われている。
- 例えば“倍精度”は 10 進 16 桁の精度をもつ

$$1 + 0.0000000000000001 = 1$$

- 結合法則は必ずしも成り立たない。

$$a + (b + c) \neq (a + b) + c$$

コンピュータでの実数演算はどうするか

コンピュータは有限なので実数の一部しか取り扱うことができないため、近似的な取り扱いをする

- 浮動小数点数: コンピュータ上での実数の近似表現の一つ。

$$1.01010101 \times 2^3$$

- とても実用的で、よく使われている。
- 例えば“倍精度”は 10 進 16 桁の精度をもつ

$$1 + 0.0000000000000001 = 1$$

- 結合法則は必ずしも成り立たない。

$$a + (b + c) \neq (a + b) + c$$

コンピュータでの実数演算はどうするか

コンピュータは有限なので実数の一部しか取り扱うことができないため、近似的な取り扱いをする

- 浮動小数点数: コンピュータ上での実数の近似表現の一つ。

$$1.01010101 \times 2^3$$

- とても実用的で、よく使われている。
- 例えば“倍精度”は 10 進 16 桁の精度をもつ

$$1 + 0.0000000000000001 = 1$$

- 結合法則は必ずしも成り立たない。

$$a + (b + c) \neq (a + b) + c$$

コンピュータでの実数演算はどうするか

コンピュータは有限なので実数の一部しか取り扱うことができないため、近似的な取り扱いをする

- 浮動小数点数: コンピュータ上での実数の近似表現の一つ。

$$1.01010101 \times 2^3$$

- とても実用的で、よく使われている。
- 例えば“倍精度”は 10 進 16 桁の精度をもつ

$$1 + 0.0000000000000001 = 1$$

- 結合法則は必ずしも成り立たない。

$$a + (b + c) \neq (a + b) + c$$

コンピュータでの実数演算はどうするか

コンピュータは有限なので実数の一部しか取り扱うことができないため、近似的な取り扱いをする

- 浮動小数点数: コンピュータ上での実数の近似表現の一つ。

$$1.01010101 \times 2^3$$

- とても実用的で、よく使われている。
- 例えば“倍精度”は 10 進 16 桁の精度をもつ

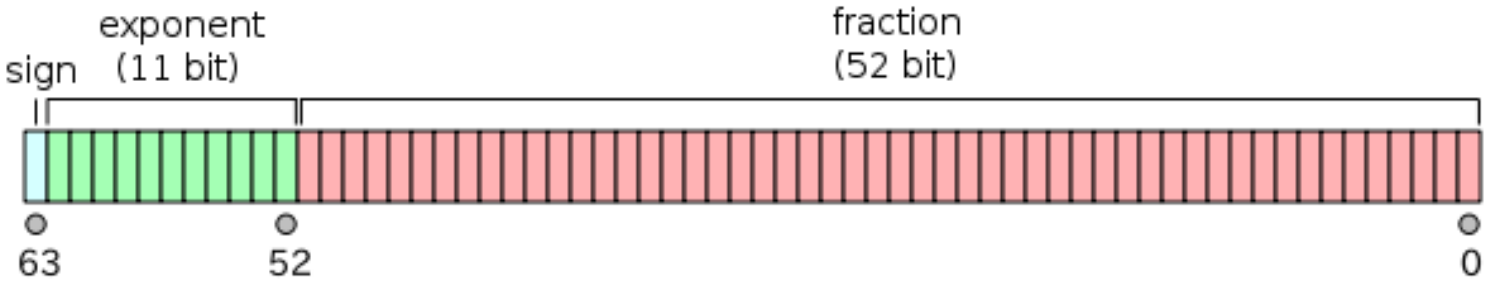
$$1 + 0.0000000000000001 = 1$$

- 結合法則は必ずしも成り立たない。

$$a + (b + c) \neq (a + b) + c$$

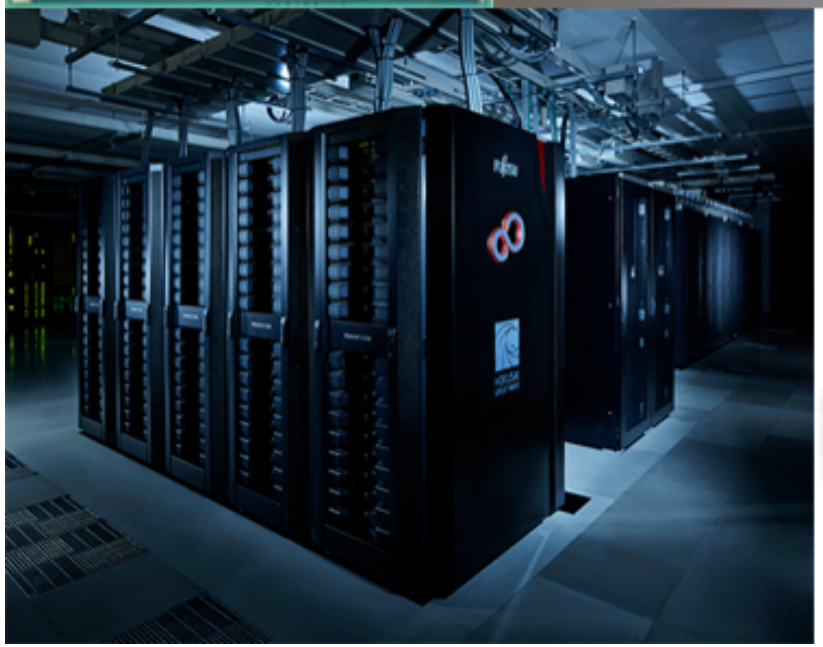
コンピュータでの数の取扱い:倍精度

- “754-2008 IEEE Standard for Floating-Point Arithmetic”
- binary64 (倍精度) フォーマットは 10 進 16 桁の有効桁がある



- ほとんどののコンピュータがこれを使っている。
- 1 秒間に 1 回浮動小数点数が計算できること=Floating point operation per second
- $\mathrm{G} = 10^9$, $\mathrm{T} = 10^{12}$, $\mathrm{P} = 10^{15}$
- 速さ:Core i7 (Haswell, 8 cores, 3.0GHz): ~384GFLOPS; NVIDIA K80 ~2.91TFLOPS, 京コンピュータ ~ 10PFLOPS), HOKUSAI (1PFlops)

RIKEN



RIKEN

コンピュータでの線形代数

自作するのは難しいことがある

線形代数の教科書に載っているやり方をそのままコンピュータに載せると...

- 線形連立一次方程式をガウスの消去法でそのまま解く。
- 行列-行列の積を求める。→ コンピュータの構造をある程度理解しないと、そのままでは大変遅い。
- クラメールの公式で線形連立一次方程式を解く。
- 行列式を求める。→ 誤差が大きくなる。行列式は通常直接求めない。
- 結果がおかしい: 収束しない, 0 で割った...
  → バグを突き止めるのは難しい時がある

コンピュータに合った計算方法が必要

自作するのは難しいことがある

線形代数の教科書に載っているやり方をそのままコンピュータに載せると...

- 線形連立一次方程式をガウスの消去法でそのまま解く。
- 行列-行列の積を求める。→ コンピュータの構造をある程度理解しないと、そのままでは大変遅い。
- クラメールの公式で線形連立一次方程式を解く。
- 行列式を求める。→ 誤差が大きくなる。行列式は通常直接求めない。
- 結果がおかしい: 収束しない, 0 で割った...
  → バグを突き止めるのは難しい時がある

コンピュータに合った計算方法が必要

RIKEN

自作するのは難しいことがある

線形代数の教科書に載っているやり方をそのままコンピュータに載せると...

- 線形連立一次方程式をガウスの消去法でそのまま解く。
- 行列-行列の積を求める。→ コンピュータの構造をある程度理解しないと、そのままでは大変遅い。
- クラメールの公式で線形連立一次方程式を解く。
- 行列式を求める。→ 誤差が大きくなる。行列式は通常直接求めない。
- 結果がおかしい: 収束しない, 0 で割った...
  → バグを突き止めるのは難しい時がある

コンピュータに合った計算方法が必要

自作するのは難しいことがある

線形代数の教科書に載っているやり方をそのままコンピュータに載せると...

- 線形連立一次方程式をガウスの消去法でそのまま解く。
- 行列-行列の積を求める。→ コンピュータの構造をある程度理解しないと、そのままでは大変遅い。
- クラメールの公式で線形連立一次方程式を解く。
- 行列式を求める。→ 誤差が大きくなる。行列式は通常直接求めない。
- 結果がおかしい: 収束しない, 0 で割った...
  → バグを突き止めるのは難しい時がある

コンピュータに合った計算方法が必要

RIKEN

自作するのは難しいことがある

線形代数の教科書に載っているやり方をそのままコンピュータに載せると...

- 線形連立一次方程式をガウスの消去法でそのまま解く。
- 行列-行列の積を求める。→ コンピュータの構造をある程度理解しないと、そのままでは大変遅い。
- クラメールの公式で線形連立一次方程式を解く。
- 行列式を求める。→ 誤差が大きくなる。行列式は通常直接求めない。
- 結果がおかしい: 収束しない, 0 で割った...
  → バグを突き止めるのは難しい時がある

コンピュータに合った計算方法が必要

RIKEN

自作するのは難しいことがある

線形代数の教科書に載っているやり方をそのままコンピュータに載せると...

- 線形連立一次方程式をガウスの消去法でそのまま解く。
- 行列-行列の積を求める。→ コンピュータの構造をある程度理解しないと、そのままでは大変遅い。
- クラメールの公式で線形連立一次方程式を解く。
- 行列式を求める。→ 誤差が大きくなる。行列式は通常直接求めない。
- 結果がおかしい: 収束しない, 0 で割った...
  → バグを突き止めるのは難しい時がある

コンピュータに合った計算方法が必要

自作するのは難しいことがある

- そんなこと言っても自分はそこまで詳しくないし、便利なプログラムはすでに無いの？

  あります。BLAS, LAPACK をつかいましょう

- コンピュータの仕組みをにあったやり方とはどうするのか？

  来週やります

- 今回は「とりあえず」使って見る、ということをやります。

自作するのは難しいことがある

- そんなこと言っても自分はそこまで詳しくないし、便利なプログラムはすでに無いの?

  ありま す。BLAS, LAPACK をつかいましょう

- コンピュータの仕組みをにあったやり方とはどうするのか?

  来週やります

- 今回は「とりあえず」使って見る、ということをやります。

自作するのは難しいことがある

- そんなこと言っても自分はそこまで詳しくないし、便利なプログラムはすでに無いの?

  あります。BLAS, LAPACK をつかいましょう

- コンピュータの仕組みをにあったやり方とはどうするのか?

  来週やります

- 今回は「とりあえず」使って見る、ということをやります。

自作するのは難しいことがある

- そんなこと言っても自分はそこまで詳しくないし、便利なプログラムはすでに無いの?

  あります。BLAS, LAPACK をつかいましょう

- コンピュータの仕組みをにあったやり方とはどうするのか?

  来週やります

- 今回は「とりあえず」使って見る、ということをやります。

自作するのは難しいことがある

- そんなこと言っても自分はそこまで詳しくないし、便利なプログラムはすでに無いの?

  あります。BLAS, LAPACK をつかいましょう

- コンピュータの仕組みをにあったやり方とはどうするのか?

  来週やります

- 今回は「とりあえず」使って見る、ということをやります。

自作するのは難しいことがある

- そんなこと言っても自分はそこまで詳しくないし、便利なプログラムはすでに無いの?

  あります。BLAS, LAPACK をつかいましょう

- コンピュータの仕組みをにあったやり方とはどうするのか?

  来週やります

- 今回は「とりあえず」使って見る、ということをやります。

BLAS, LAPACK: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するなら BLAS+LAPACK を使いましょう。
- 品質、信頼性がとても高いです。
- 無料で入手出来ます。
- 高速版 (機種、OS による) がある場合もあります。
- ただし密行列向けです (疎行列はサポートされてません)。

BLAS, LAPACK は人類の宝!

BLAS, LAPACK: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するなら BLAS+LAPACK を使いましょう。
- 品質、信頼性がとても高いです。
- 無料で入手出来ます。
- 高速版 (機種、OS による) がある場合もあります。
- ただし密行列向けです (疎行列はサポートされてません)。

BLAS, LAPACK は人類の宝!

BLAS, LAPACK: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するなら BLAS+LAPACK を使いましょう。
- 品質、信頼性がとても高いです。
- 無料で入手出来ます。
- 高速版 (機種、OS による) がある場合もあります。
- ただし密行列向けです (疎行列はサポートされてません)。

BLAS, LAPACK は人類の宝!

BLAS, LAPACK: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するなら BLAS+LAPACK を使いましょう。
- 品質、信頼性がとても高いです。
- 無料で入手出来ます。
- 高速版 (機種、OS による) がある場合もあります。
- ただし密行列向けです (疎行列はサポートされてません)。

BLAS, LAPACK は人類の宝!

BLAS, LAPACK: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するなら BLAS+LAPACK を使いましょう。
- 品質、信頼性がとても高いです。
- 無料で入手出来ます。
- 高速版 (機種、OS による) がある場合もあります。
- ただし密行列向けです (疎行列はサポートされてません)。

BLAS, LAPACK は人類の宝!

BLAS, LAPACK: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するなら BLAS+LAPACK を使いましょう。
- 品質、信頼性がとても高いです。
- 無料で入手出来ます。
- 高速版 (機種、OS による) がある場合もあります。
- ただし密行列向けです (疎行列はサポートされてません)。

BLAS, LAPACK は人類の宝!

BLAS, LAPACK: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するなら BLAS+LAPACK を使いましょう。
- 品質、信頼性がとても高いです。
- 無料で入手出来ます。
- 高速版 (機種、OS による) がある場合もあります。
- ただし密行列向けです (疎行列はサポートされてません)。

BLAS, LAPACK は人類の宝!

BLAS とは?

- BLAS は Basic Linear Algebra Subprograms の略
- 基礎的な線形代数の「サブ」プログラム
  - ベクトル-ベクトルの内積
  - 行列-ベクトル積
  - 行列-行列積
- FORTRAN77 でさまざまなルーチンの仕様を提供している。
- 参照実装の形で提供されている (reference BLAS)
  - BLAS のルーチンを「ブロック」にしてより高度なことをする。
  - この実装を「お手本」とする
  - とても美しいコード!
  - 高速版もある。

http://www.netlib.org/blas

RIKEN

BLAS とは?

- BLAS は Basic Linear Algebra Subprograms の略
- 基礎的な線形代数の「サブ」プログラム
  - ベクトル-ベクトルの内積
  - 行列-ベクトル積
  - 行列-行列積
- FORTRAN77 でさまざまなルーチンの仕様を提供している。
- 参照実装の形で提供されている (reference BLAS)
  - BLAS のルーチンを「ブロック」にしてより高度なことをする。
  - この実装を「お手本」とする
  - とても美しいコード!
  - 高速版もある。

http://www.netlib.org/blas

RIKEN

BLAS とは?

- BLAS は Basic Linear Algebra Subprograms の略
- 基礎的な線形代数の「サブ」プログラム
  - ベクトル-ベクトルの内積
  - 行列-ベクトル積
  - 行列-行列積
- FORTRAN77 でさまざまなルーチンの仕様を提供している。
- 参照実装の形で提供されている (reference BLAS)
  - BLAS のルーチンを「ブロック」にしてより高度なことをする。
  - この実装を「お手本」とする
  - とても美しいコード!
  - 高速版もある。

http://www.netlib.org/blas

RIKEN

BLAS とは?

- BLAS は Basic Linear Algebra Subprograms の略
- 基礎的な線形代数の「サブ」プログラム
  - ベクトル-ベクトルの内積
  - 行列-ベクトル積
  - 行列-行列積
- FORTRAN77 でさまざまなルーチンの仕様を提供している。
- 参照実装の形で提供されている (reference BLAS)
  - BLAS のルーチンを「ブロック」にしてより高度なことをする。
  - この実装を「お手本」とする
  - とても美しいコード!
  - 高速版もある。

http://www.netlib.org/blas

RIKEN

BLAS とは?

- BLAS は Basic Linear Algebra Subprograms の略
- 基礎的な線形代数の「サブ」プログラム
  - ベクトル-ベクトルの内積
  - 行列-ベクトル積
  - 行列-行列積
- FORTRAN77 でさまざまなルーチンの仕様を提供している。
- 参照実装の形で提供されている (reference BLAS)
  - BLAS のルーチンを「ブロック」にしてより高度なことをする。
  - この実装を「お手本」とする
  - とても美しいコード!
  - 高速版もある。

http://www.netlib.org/blas

RIKEN

Level 1 BLAS

BLAS には Level 1, 2, 3 と三種類のものがある。
Level 1:ベクトル-ベクトル演算 (+そのほか) のルーチン群

- ベクトルの加算 (DAXPY),

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

- 内積計算 (DDOT)

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

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

Level 2 BLAS

BLAS には Level 1, 2, 3 と三種類のものがある。
Level 2:行列-ベクトル演算ルーチン群

- 行列-ベクトル積 (DGEMV)

$$\boldsymbol{y} \leftarrow \alpha A\boldsymbol{x} + \beta\boldsymbol{y}, \tag{3}$$

- 上三角行列の連立一次方程式を解く (DTRSV)

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

など 25 種類あり, 同じように 4 通りの組み合わせがある。

Level 3 BLAS

BLAS には Level 1, 2, 3 と三種類のものがある。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`),

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

- 行列-行列積 `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 種類ある。

BLAS の命名規則とルーチン

型:単精度、倍精度、単精度複素数、倍精度複素数で接頭辞 “s”, “d”, “c”, “z” がつく。

LEVEL1 BLAS

zrotg	zdcal	drotg	drot	drotm	zdrot	zswap
dswap	zdscal	dscal	zcopy	dcopy	zaxpy	daxpy
ddot	zdotc	zdotu	dznrm2	dnrm2	dasum	izasum
idamax	dzabs1					

LEVEL2 BLAS

zgemv	dgemv	zgbmv	dgbmv	zhemv	zhbmv	zhpmv	dsymv
dsbmv	ztrmv	zgemv	dgemv	zgbmv	dgemv	zhemv	zhbmv
zhpmv	dsymv	dsbmv	dspmv	ztrmv	dtrmv	ztbmv	ztpmv
dtpmv	ztrsv	dtrsv	ztbsv	dtbsv	ztpsv	dger	zgeru
zgerc	zher	zhpr	zher2	zhpr2	dsyr	dspr	dsyr2
dspr2							

LEVEL3 BLAS

zgemm	dgemm	zsymm	dsymm	zhemm	zsyrk	dsyrk	zherk
zsyr2k	dsyr2k	zher2k	ztrmm	dtrmm	ztrsm	dtrsm	

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.
- 下請けルーチン群も提供する: 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。
- 品質保証も非常に精密かつ系統的で、信頼がおける。
- パソコンからスーパーコンピュータまで様々な CPU、OS 上で動く。
- Fortran 90 で書かれ、3.4.2 は 1600 以上のルーチンからなっている。
- web サイトはなんと 1 億 1600 万ヒットである!

http://www.netlib.org/lapack

RIKEN

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.

- 物理、化学では Gaussian, Gamess, ADF, VASP
- 線形計画問題の CPLEX, NUOPT, GLPK など..
- 高級言語からも利用可能 Ruby, Python, Perl, Java, C, Mathematica, Maple, Matlab, R, octave, SciLab

Top 500

Top 500: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。

http://www.top500.org/

BLAS, LAPACK の現状:高速な BLAS, LAPACK について

Reference BLAS はある意味仕様書そのままなので、非常に低速である。メモリの階層構造などは非常に意識して書かれているが、CPU に最適化は、各々がやる、というスタンスである。

- GotoBLAS2: 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, LAPACK バージョン 3.1.1 も含みさらに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 SandyBridge 以降のプロセッサには対応せず。
- OpenBLAS: Zhang Xianyi 氏が GotoBLAS2 の開発を引き継いだ。開発はアクティブで SandyBridge 以降のプロセッサにも対応している。また、ICT Loongson-3A, 3B にも対応。
- Intel MKL: Intel が開発している加速された BLAS および LAPACK。2012 年から後藤氏が Intel にいったため事実上 GotoBLAS2 の血も入っているため、Intel 系では最速と思われる。

BLAS, LAPACK の現状:高速な BLAS, LAPACK について

- ATLAS: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。それまでの 2001 年より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。ハンドチューニングした BLAS よりは数%から数 10%低速程度
- GPU 向け BLAS, LAPACK: 近年 CPU の性能が頭打ちになっているが, 近年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視されている。
- 並列版 BLAS, LAPACK: 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。

BLAS, LAPACK を使う上での注意点:Column major or Row major

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 column major である。

FORTRAN や、Matlab, octave は column major である。

BLAS, LAPACKを使う上での注意点:Column major or Row major

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

$$1, 2, 3, 4, 5, 6$$

のように格納されるのが row major である。C, C++では普通 row major である。

C/C++から BLAS, LAPACK を呼び出すときは、行列の major に注意!!

BLAS, LAPACK を使う上での注意点:leading dimension

行列の部分行列を使うと便利なことがある (実際 LU 分解, Chokesky 分解などで LAPACK では多用されている)。そのために, “leading dimension” が設定されている。BLAS, LAPACK の LDA, LDB などの引数. FORTRAN では、$A(i + j * m)$ でなくて,

$$A(i + j * lda)$$

要素にアクセスしなければならない。 下図で $M \times N$ の行列 A は $LDA \times N$ の行列の部分行列となっている。

BLAS, LAPACK を使う上での注意点:配列は 0 か 1 どちらから始まるか?

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 では $x[i-1]$ 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 では column major として $A[i-1+(j-1)*lda]$ とする。

などである。

BLAS, LAPACK を使う上での注意点:環境依存が激しい

- OS のバージョン、どの最適化 BLAS を使うかなどによって大きくやり方がかわる。
  - コンパイラ、ライブラリのインストールの仕方
  - どのように BLAS をコールするか (CBLAS を使うべきか? C から FORTRAN のサブルーチンは公式には呼べなかった)
  - Fortran のランタイムをどうリンクするか。
  - integer は 32bit か 64bit か? (64bit BLAS があるか否か)
- GPU などを使うとなると、さらに複雑になる。
- 実行環境を整えるのは、Linux が一番楽、MacOSX が次、Windows が一番難しい。
- 今回は Ubuntu 12.04 x86 を使った。

BLAS、LAPACKを使ってみる

Ubuntu 12.04 デスクトップ版で実際に BLAS, LAPACK を実際に使ってみる。C++から

- 行列-行列積
- 対称行列の対角化

を行う。

BLAS、LAPACK のインストール

Ubuntu 12.04 で次のようにすると、BLAS、LAPACK の開発環境が整う。

```
$ sudo apt-get install gfortran g++ libblas-dev liblapack-dev
パッケージリストを読み込んでいます... 完了
依存関係ツリーを作成しています
状態情報を読み取っています... 完了
...
```

成功したら二回目の実行で

```
$ sudo apt-get install gfortran g++ libblas-dev liblapack-dev
...
g++ はすでに最新バージョンです。
gfortran はすでに最新バージョンです。
libblas-dev はすでに最新バージョンです。
liblapack-dev はすでに最新バージョンです。
アップグレード: 0 個、新規インストール: 0 個、削除: 0 個、保留: 172 個。
```

となるはず。

行列-行列の積

行列-行列積 DGEMM を使ってみよう. ここでは、

$$A = \begin{pmatrix} 1 & 8 & 3 \\ 2 & 10 & 8 \\ 9 & -5 & -1 \end{pmatrix} \quad B = \begin{pmatrix} 9 & 8 & 3 \\ 3 & 11 & 2.3 \\ -8 & 6 & 1 \end{pmatrix} \quad C = \begin{pmatrix} 3 & 3 & 1.2 \\ 8 & 4 & 8 \\ 6 & 1 & -2 \end{pmatrix}$$

$\alpha = 3, \beta = -2$ として,

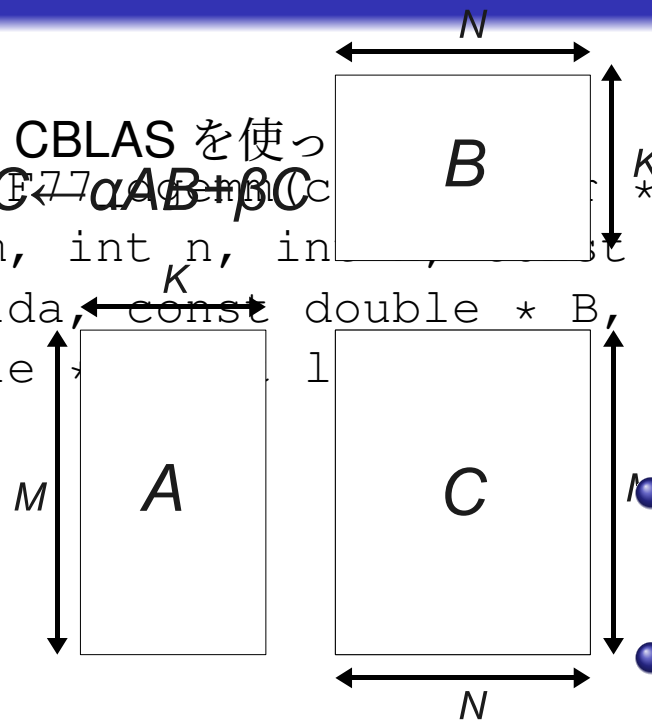
$$C \leftarrow \alpha AB + \beta C$$

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。

行列-行列の積:DGEMM の詳細



今回は CBLAS を使っ

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

```
void F77_dgemm(c                *transa, const char *transb,
int m, int n, in                t double * alpha, const double *A,
int lda, const double * B, int ldb, const double *beta,
double *                   l
```

- “transa”, “transb”, “transc” で行列を転置するか否かを指定。
- A, B, C は行列への Row major の配列、またはポインタ
- M, N, K は行列の次元。左図参照
- alpha, beta は行列積に対する掛けるスカラー

行列-行列の積のリストI

```
#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;
```

行列-行列の積のリスト II

```
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;
```

行列-行列の積のコンパイルと実行

先ほどのリストを”dgemm_demo.cpp” などと保存する。

```
$ g++ dgemm_demo.cpp -o dgemm_demo -lblas -lapack
```

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。Octave や Matlab にこの結果をそのままコピー&ペースとすれば答えをチェックできるようにしてある。

```
$ ./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
```

LAPACK 実習:行列の固有ベクトル、固有値を求める:DSYEV

3×3 の実対称行列

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

の固有ベクトル、固有値を求めよう。これらは三つあり、

$$A\boldsymbol{v}_i = \lambda_i \boldsymbol{v}_i \quad (i = 1, 2, 3)$$

という関係式が成り立つ。固有値 $\lambda_1, \lambda_2, \lambda_3$ は、

$$-0.40973, 1.57715, 10.83258$$

で、 固有ベクトル $\boldsymbol{v}_1, \boldsymbol{v}_2, \boldsymbol{v}_3$ は、

$$\begin{aligned} \boldsymbol{v}_1 &= (-0.914357, 0.216411, 0.342225) \\ \boldsymbol{v}_2 &= (0.040122, -0.792606, 0.608413) \\ \boldsymbol{v}_3 &= (0.402916, 0.570037, 0.716042) \end{aligned}$$

となる。

行列の固有ベクトル、固有値を求める DSYEV 詳細

今回は Fortran を直接よんでみる

```
dsyev_f77(const char *jobz, const char *uplo, int *n,
double *A, int *lda, double *w, double *work,
int *lwork, int *info);
```

- jobz:固有値、固有ベクトルが必要か、固有値だけでよいか指定。
- uplo:行列の上三角、下三角を使うか。
- A, lda:行列 A とその leading dimension
- w:固有値を返す配列 (昇順)
- work, lwork:ワーク領域への配列またはポインタ、とそのサイズ
- info: =0 正常終了。<0: INFO=-i では i 番目の引数が不適当。>0: INFO=i 収束せず。

行列の対角化のリスト

```
#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;
```

対称行列の対角化のコンパイルと実行

先ほどのリストを”eigenvalue_demo.cpp” などと保存する。次に

```
$ g++ eigenvalue_demo.cpp -o eigenvalue_demo -lblas
          -lapack -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。同様に Octave や Matlab にこの結果をそのままコピー&ペーストとすれば答えをチェックできるようにしてある。

```
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
```

まとめと次回予告

まとめ

- 線形代数の重要性、歴史についてのべた。
- BLAS, LAPACK について簡単な説明をした。
- BLAS, LAPACK について簡単な使い方を示した。

次回予告

- コンピュータの簡単なしくみ。
- BLAS, LAPACK を高速につかう。