高精度線形代数演算ライブラリ MPLAPACK の紹介とデモ

中田真秀

理化学研究所, 情報基盤センター 2F

2019/3/18

MPLAPACK 0.8.0: 高精度版の BLAS and LAPACK

http://mplapack.sourceforge.net/

https://github.com/nakatamaho/mplapack

中田真秀 @ 理研

- MPLAPACK: multiple precision version of BLAS and LAPACK; 改名しました
- ビルディングブロック、参照実装、API の提供
- Version 0.8.0 (2012/12/25); 状況: MBLAS 完成, and 100 MLAPACK ルーチン.
- マルチプラットフォーム:Linux/BSD/Mac/Win...
- 5 つの高精度ライブラリをサポート: GMP, MPFR, DD, QD, double
- C++で書いてある: 簡単、高速なプログラミング
- フリーソフトウェア: 2 条項 BSD ライセンス: 改変、再配布が自由にできる。

あらまし

あらまし

あらまし

あらまし

より高精度への要求: ペタ、エクサフロップスに向かって

- エクサスケールでは数値計算誤差が大きな問題になってくる可能性がある
  - 一週間で 10^{23} 回の計算: 微視から巨視までのスケール
- IEEE 754 double (10 進数 16 桁) はもう充分でない。→ 多倍長精度 (multiple precision,MP) が必要。
- 反復法 (esp. Krylov 部分空間法等) は高精度だと収束したり、反復回数が減ったりする。
- 半正定値計画法: 条件数は最適解で発散する。
- LAPACK チームの調査: 16%のユーザーがよくまたは時々高精度を使っている。

高精度版線形代数ライブラリの開発は重要

より高精度への要求: ペタ、エクサフロップスに向かって

- エクサスケールでは数値計算誤差が大きな問題になってくる可能性がある
  - 一週間で 10^{23} 回の計算: 微視から巨視までのスケール
- IEEE 754 double (10 進数 16 桁) はもう充分でない。→ 多倍長精度 (multiple precision,MP) が必要。
- 反復法 (esp. Krylov 部分空間法等) は高精度だと収束したり、反復回数が減ったりする。
- 半正定値計画法: 条件数は最適解で発散する。
- LAPACK チームの調査: 16%のユーザーがよくまたは時々高精度を使っている。

高精度版線形代数ライブラリの開発は重要

より高精度への要求: ペタ、エクサフロップスに向かって

- エクサスケールでは数値計算誤差が大きな問題になってくる可能性がある
  - 一週間で 10^{23} 回の計算: 微視から巨視までのスケール
- IEEE 754 double (10 進数 16 桁) はもう充分でない。→ 多倍長精度 (multiple precision,MP) が必要。
- 反復法 (esp. Krylov 部分空間法等) は高精度だと収束したり、反復回数が減ったりする。
- 半正定値計画法: 条件数は最適解で発散する。
- LAPACK チームの調査: 16%のユーザーがよくまたは時々高精度を使っている。

高精度版線形代数ライブラリの開発は重要

より高精度への要求: ペタ、エクサフロップスに向かって

- エクサスケールでは数値計算誤差が大きな問題になってくる可能性がある
  - 一週間で 10^{23} 回の計算: 微視から巨視までのスケール
- IEEE 754 double (10 進数 16 桁) はもう充分でない。→ 多倍長精度 (multiple precision,MP) が必要。
- 反復法 (esp. Krylov 部分空間法等) は高精度だと収束したり、反復回数が減ったりする。
- 半正定値計画法: 条件数は最適解で発散する。
- LAPACK チームの調査: 16%のユーザーがよくまたは時々高精度を使っている。

高精度版線形代数ライブラリの開発は重要

より高精度への要求: ペタ、エクサフロップスに向かって

- エクサスケールでは数値計算誤差が大きな問題になってくる可能性がある
  - 一週間で 10^{23} 回の計算: 微視から巨視までのスケール
- IEEE 754 double (10 進数 16 桁) はもう充分でない。→ 多倍長精度 (multiple precision,MP) が必要。
- 反復法 (esp. Krylov 部分空間法等) は高精度だと収束したり、反復回数が減ったりする。
- 半正定値計画法: 条件数は最適解で発散する。
- LAPACK チームの調査: 16%のユーザーがよくまたは時々高精度を使っている。

高精度版線形代数ライブラリの開発は重要

より高精度への要求: ペタ、エクサフロップスに向かって

- エクサスケールでは数値計算誤差が大きな問題になってくる可能性がある
  - 一週間で 10^{23} 回の計算: 微視から巨視までのスケール
- IEEE 754 double (10 進数 16 桁) はもう充分でない。→ 多倍長精度 (multiple precision,MP) が必要。
- 反復法 (esp. Krylov 部分空間法等) は高精度だと収束したり、反復回数が減ったりする。
- 半正定値計画法: 条件数は最適解で発散する。
- LAPACK チームの調査: 16%のユーザーがよくまたは時々高精度を使っている。

高精度版線形代数ライブラリの開発は重要

より高精度への要求: ペタ、エクサフロップスに向かって

- エクサスケールでは数値計算誤差が大きな問題になってくる可能性がある
  - 一週間で 10^{23} 回の計算: 微視から巨視までのスケール
- IEEE 754 double (10 進数 16 桁) はもう充分でない。→ 多倍長精度 (multiple precision,MP) が必要。
- 反復法 (esp. Krylov 部分空間法等) は高精度だと収束したり、反復回数が減ったりする。
- 半正定値計画法: 条件数は最適解で発散する。
- LAPACK チームの調査: 16%のユーザーがよくまたは時々高精度を使っている。

高精度版線形代数ライブラリの開発は重要

丸め誤差の影響をを高精度計算で解決する

- 浮動小数点: 実数の計算機上での近似的取扱い。
- 丸め誤差が演算ごとに起こりうる
- 誤差への (一つの) 解決法: 高精度計算

高精度計算は丸め誤差に対する brute force な解決方法

丸め誤差の影響をを高精度計算で解決する

- 浮動小数点: 実数の計算機上での近似的取扱い。
- 丸め誤差が演算ごとに起こりうる
- 誤差への (一つの) 解決法: 高精度計算

高精度計算は丸め誤差に対する brute force な解決方法

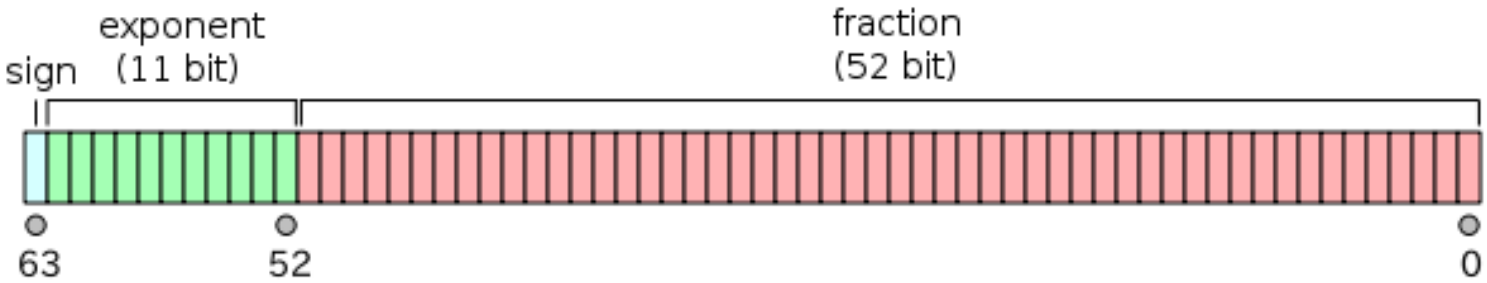
丸め誤差の影響をを高精度計算で解決する

- 浮動小数点: 実数の計算機上での近似的取扱い。
- 丸め誤差が演算ごとに起こりうる
- 誤差への (一つの) 解決法: 高精度計算

高精度計算は丸め誤差に対する brute force な解決方法

IEEE 754: Standard for Binary Floating-Point Arithmetic

- 754-2008 IEEE Standard for Floating-Point Arithmetic
- もっとも使われているし高速 (on Core i7 10 cores broadwell: ~560GFlops; AMD Fire pro S9170 ~2.62 TFLOPS)
- 10 進数 16 桁 16 binary64 (いわゆる倍精度) 64bit

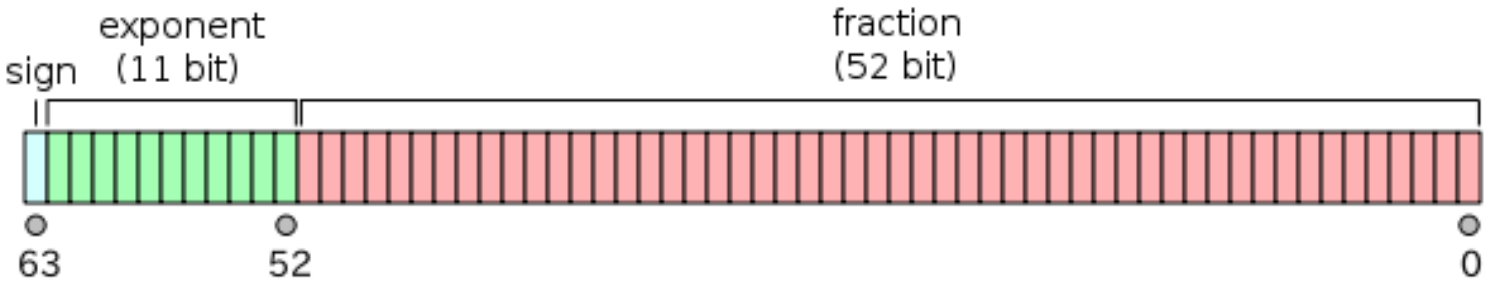


- $a = \pm\left(\frac{1}{2} + \frac{d_2}{2^2} + \frac{d_3}{2^3} + \cdots + \frac{d_{52}}{2^{52}}\right) \times 2^e$, $d = 0$ or 1, $e = -1022 \sim 1023$

(Partially taken from Wikipedia: http://en.wikipedia.org/wiki/IEEE_754-2008).

IEEE 754: Standard for Binary Floating-Point Arithmetic

- 754-2008 IEEE Standard for Floating-Point Arithmetic
- もっとも使われているし高速 (on Core i7 10 cores broadwell: ~560GFlops; AMD Fire pro S9170 ~2.62 TFLOPS)
- 10 進数 16 桁 16 binary64 (いわゆる倍精度) 64bit

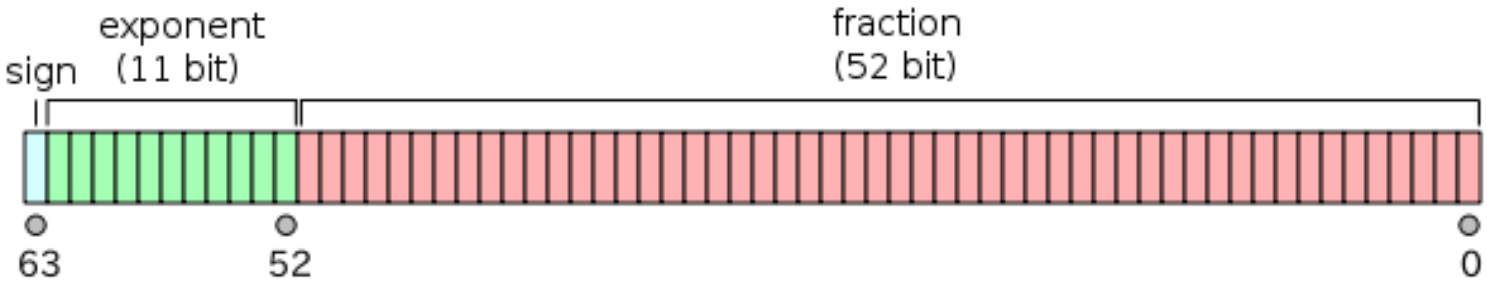


- $a = \pm\left(\frac{1}{2} + \frac{d_2}{2^2} + \frac{d_3}{2^3} + \cdots + \frac{d_{52}}{2^{52}}\right) \times 2^e$, $d = 0$ or 1, $e = -1022 \sim 1023$

(Partially taken from Wikipedia: http://en.wikipedia.org/wiki/IEEE_754-2008).

IEEE 754: Standard for Binary Floating-Point Arithmetic

- 754-2008 IEEE Standard for Floating-Point Arithmetic
- もっとも使われているし高速 (on Core i7 10 cores broadwell: ~560GFlops; AMD Fire pro S9170 ~2.62 TFLOPS)
- 10 進数 16 桁 16 binary64 (いわゆる倍精度) 64bit

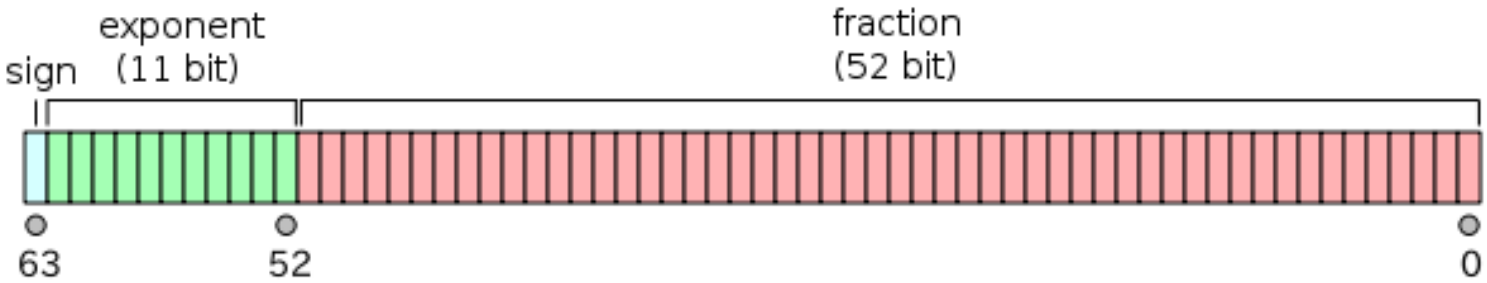


- $a = \pm\left(\frac{1}{2} + \frac{d_2}{2^2} + \frac{d_3}{2^3} + \cdots + \frac{d_{52}}{2^{52}}\right) \times 2^e, d = 0 \text{ or } 1,$
  $e = -1022 \sim 1023$

(Partially taken from Wikipedia: http://en.wikipedia.org/wiki/IEEE_754-2008).

IEEE 754: Standard for Binary Floating-Point Arithmetic

- 754-2008 IEEE Standard for Floating-Point Arithmetic
- もっとも使われているし高速 (on Core i7 10 cores broadwell: ~560GFlops; AMD Fire pro S9170 ~2.62 TFLOPS)
- 10 進数 16 桁 16 binary64 (いわゆる倍精度) 64bit

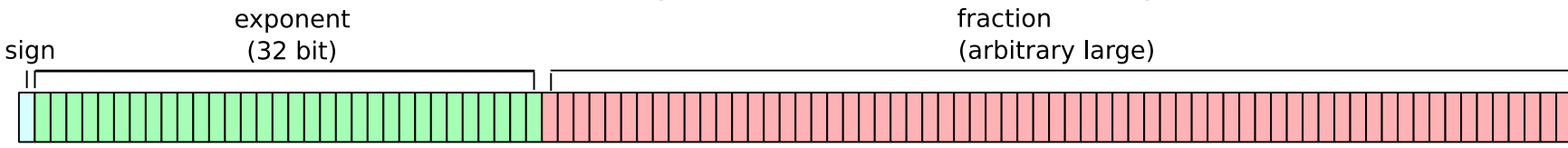


- $a = \pm\left(\frac{1}{2} + \frac{d_2}{2^2} + \frac{d_3}{2^3} + \cdots + \frac{d_{52}}{2^{52}}\right) \times 2^e$, $d = 0$ or 1, $e = -1022 \sim 1023$

(Partially taken from Wikipedia: http://en.wikipedia.org/wiki/IEEE_754-2008).

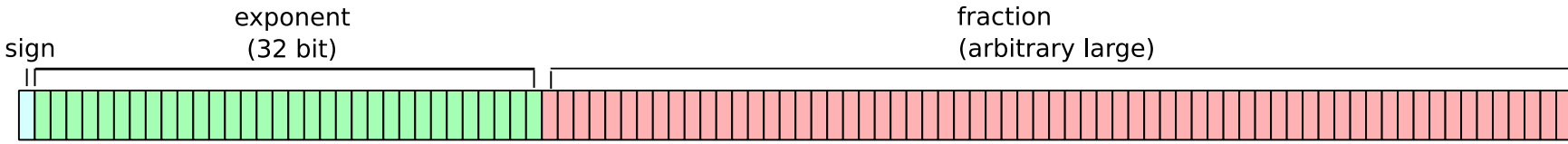
高精度計算とは

- GMP とは? GMP は自由なライブラリで、符号つき整数、有理数、浮動小数点数を扱える。
- 精度は任意に大きく選べる (32, 64 ビットの倍数)



高精度計算とは

- GMP とは? GMP は自由なライブラリで、符号つき整数、有理数、浮動小数点数を扱える。
- 精度は任意に大きく選べる (32, 64 ビットの倍数)



他の高精度演算ライブラリ

他の高精度演算ライブラリ

- 倍倍精度, 四-倍精度 (一番高速:10 から 200 倍程度しか遅くない): QD, DD ライブラリ
- binary128 本物の 4 倍精度: float128 (将来にむけた練習用?)
- 丸めモード変更のサポート (100 倍から 1000 倍程度遅い): MPFR, MPC
- とりあえずの高精度計算: GMP (MPFR, MPC などよりかは少し高速)

デマンドに従って様々な形態のライブラリが存在している。

このプロジェクトでは高精度演算ライブラリの構築は目的としていない

BLAS, LAPACK とはなにか?

- BLAS: ベクトル-ベクトル, 行列-ベクトル, 行列-行列の演算を行う。参照実装や高速な実装 (GotoBLAS2, Intel MKL, ATLAS etc.) もある。
- LAPACK: 線形連立一次方程式、固有値問題, 最小二乗法、特異値分解等を BLAS をブロックとして解く。
- デファクトスタンダードのライブラリ: 気づかずに使っていることも多々ある。
- LAPACK の web のヒット数: 90,340,774(Thu Feb 24 12:54:24 EST 20110)

BLAS, LAPACK は大変重要なライブラリ

BLAS, LAPACK とはなにか?

- BLAS: ベクトル-ベクトル, 行列-ベクトル, 行列-行列の演算を行う。参照実装や高速な実装 (GotoBLAS2, Intel MKL, ATLAS etc.) もある。
- LAPACK: 線形連立一次方程式、固有値問題, 最小二乗法、特異値分解等を BLAS をブロックとして解く。
- デファクトスタンダードのライブラリ: 気づかずに使っていることも多々ある。
- LAPACK の web のヒット数: 90,340,774(Thu Feb 24 12:54:24 EST 20110)

BLAS, LAPACK は大変重要なライブラリ

MPLAPACK 0.8.0: 高精度版の BLAS and LAPACK

http://mplapack.sourceforge.net/

- MPLAPACK: multiple precision version of BLAS and LAPACK.
- ビルディングブロック、参照実装、API の提供
- Version 0.8.0 (2010/8/20); 状況: MBLAS 完成, and 100 MLAPACK ルーチン.
- マルチプラットフォーム:Linux/BSD/Mac/Win...
- 5 つの高精度ライブラリをサポート: GMP, MPFR, DD, QD, double
- C++で書いてある: 簡単、高速なプログラミング
- フリーソフトウェア: 2 条項 BSD ライセンス: 改変、再配布が自由にできる。

MPLAPACK 0.8.0: 何ができて何ができないか

- Version 0.8.0 (2010/8/20); MBLAS 全 76 が完成, 100 MLAPACK ルーチン完成.
- MLAPACK でできること: 実, 複素対称行列対角化, LU 分解, Cholesky 分解, 条件数推定, 逆行列
- MLAPACK でまだできないこと: 非対称対角化, 固有値分解, 最小二乗法, QR 分解など..

MPLAPACK 紹介:API 提供: 命名規則

プリフィックスの変化
float, double → “R”eal,
complex, double complex → “C”complex.

- daxpy, zaxpy → Raxpy, Caxpy
- dgemm, zgemm → Rgemm, Cgemm
- dsterf, dsyev → Rsterf, Rsyev
- dzabs1, dzasum → RCabs1, RCasum

MBLAS 0.8.0 でサポートされているルーチン

LEVEL1 MBLAS

Crotg	Cscal	Rrotg	Rrot	Rrotm	CRrot	Cswap
Rswap	CRscal	Rscal	Ccopy	Rcopy	Caxpy	Raxpy
Rdot	Cdotc	Cdotu	RCnrm2	Rnrm2	Rasum	iCasum
iRamax	RCabs1	Mlsame	Mxerbla			

LEVEL2 MBLAS

Cgemv	Rgemv	Cgbmv	Rgbmv	Chemv	Chbmv	Chpmv	Rsymv
Rsbmv	Ctrmv	Cgemv	Rgemv	Cgbmv	Rgemv	Chemv	Chbmv
Chpmv	Rsymv	Rsbmv	Rspmv	Ctrmv	Rtrmv	Ctbmv	Ctpmv
Rtpmv	Ctrsv	Rtrsv	Ctbsv	Rtbsv	Ctpsv	Rger	Cgeru
Cgerc	Cher	Chpr	Cher2	Chpr2	Rsyr	Rspr	Rsyr2
Rspr2							

LEVEL3 MBLAS

Cgemm	Rgemm	Csymm	Rsymm	Chemm	Csyrk	Rsyrk	Cherk
Csyr2k	Rsyr2k	Cher2k	Ctrmm	Rtrmm	Ctrsm	Rtrsm	

MLAPACK 0.8.0 でサポートされている 100 ルーチン

Mutils	Rlamch	Rlae2	Rlaev2	Claev2	Rlassq	Classq
Rlanst	Clanht	Rlansy	Clansy	Clanhe	Rlapy2	Rlarfg
Rlapy3	Rladiv	Cladiv	Clarfg	Rlartg	Clartg	Rlaset
Claset	Rlasr	Clasr	Rpotf2	Clacgv	Cpotf2	Rlascl
Clascl	Rlasrt	Rsytd2	Chetd2	Rsteqr	Csteqr	Rsterf
Rlarf	Clarf	Rorg2l	Cung2l	Rorg2r	Cung2r	Rlarft
Clarft	Rlarfb	Clarfb	Rorgqr	Cungqr	Rorgql	Cungql
Rlatrd	Clatrd	Rsytrd	Chetrd	Rorgtr	Cungtr	Rsyev
Cheev	Rpotrf	Cpotrf	Clacrm	Rtrti2	Ctrti2	Rtrtri
Ctrtri	Rgetf2	Cgetf2	Rlaswp	Claswp	Rgetrf	Cgetrf
Rgetri	Cgetri	Rgetrs	Cgetrs	Rgesv	Cgesv	Rtrtrs
Ctrtrs	Rlasyf	Clasyf	Clahef	Clacrt	Claesy	Crot
Cspmv	Cspr	Csymv	Csyr	iCmax1	RCsum1	Rpotrs
Rposv	Rgeequ	Rlatrs	Rlange	Rgecon	Rlauu2	Rlauum
Rpotri	Rpocon					

赤字は 0.6.6 から 0.8.0 に向けての 10 個のルーチン

API の提供: コール方法の違い

違い: call by value / call by reference

MBLAS/MLAPACK:

```
Rgemm("n", "n", n, n, n, alpha, A, n, B, n, beta, C, n);
Rgetrf(n, n, A, n, ipiv, &info);
Rgetri(n, A, n, ipiv, work, lwork, &info);
Rsyev("V", "U", n, A, n, w, work, &lwork, &info);
```

BLAS/LAPACK:

```
dgemm_f77("N", "N", &n, &n, &n, &One, A, &n, A, &n, &Zero, C,
dgetri_f77(&n, A, &n, ipiv, work, &lwork, &info);
```

プログラミングモデル

- 数値の型: INTEGER, REAL, COMPLEX, LOGICAL.
- 高精度ライブラリを “typedef” で入れ替え REAL → mpf_class, qd_real, dd_real etc.
- 初等関数も必要 (log, sin etc); 倍精度程度の精度で充分
- 現在サポートされている高精度計算ライブラリ: GMP, MPFR, QD (DD), double
- 中間的な関数で高精度計算ライブラリの違いを吸収
- ほぼ C++の “double” と同じ感覚でプログラミングできる。

MPLAPACK でなぜいくつも高精度演算ライブラリをサポートするか

- 倍倍精度, 四-倍精度 (一番高速:10 から 200 倍程度しか遅くない): QD, DD ライブラリ
- binary128 本物の 4 倍精度: float128 (将来にむけた練習用?)
- 丸めモード変更のサポート (100 倍から 1000 倍程度遅い): MPFR, MPC
- とりあえずの高精度計算: GMP (MPFR, MPC などよりかは少し高速)
- double (普通の倍精度): デバッグ用

計算デマンドに従って様々な形態のライブラリが存在している。

MBLAS のコードの例

Caxpy: axpy の複素数版から抜粋

```
void Caxpy(INTEGER n, COMPLEX ca, COMPLEX * cx, INTEGER incx, COMPLEX
{
    REAL Zero = 0.0;
    if (n <= 0)
        return;
    if (RCabs1(ca) == Zero)
        return;
    INTEGER ix = 0;
    INTEGER iy = 0;
    if (incx < 0)
        ix = (-n + 1) * incx;
    if (incy < 0)
        iy = (-n + 1) * incy;
    for (INTEGER i = 0; i < n; i++) {
        cy[iy] = cy[iy] + ca * cx[ix];
        ix = ix + incx;
```

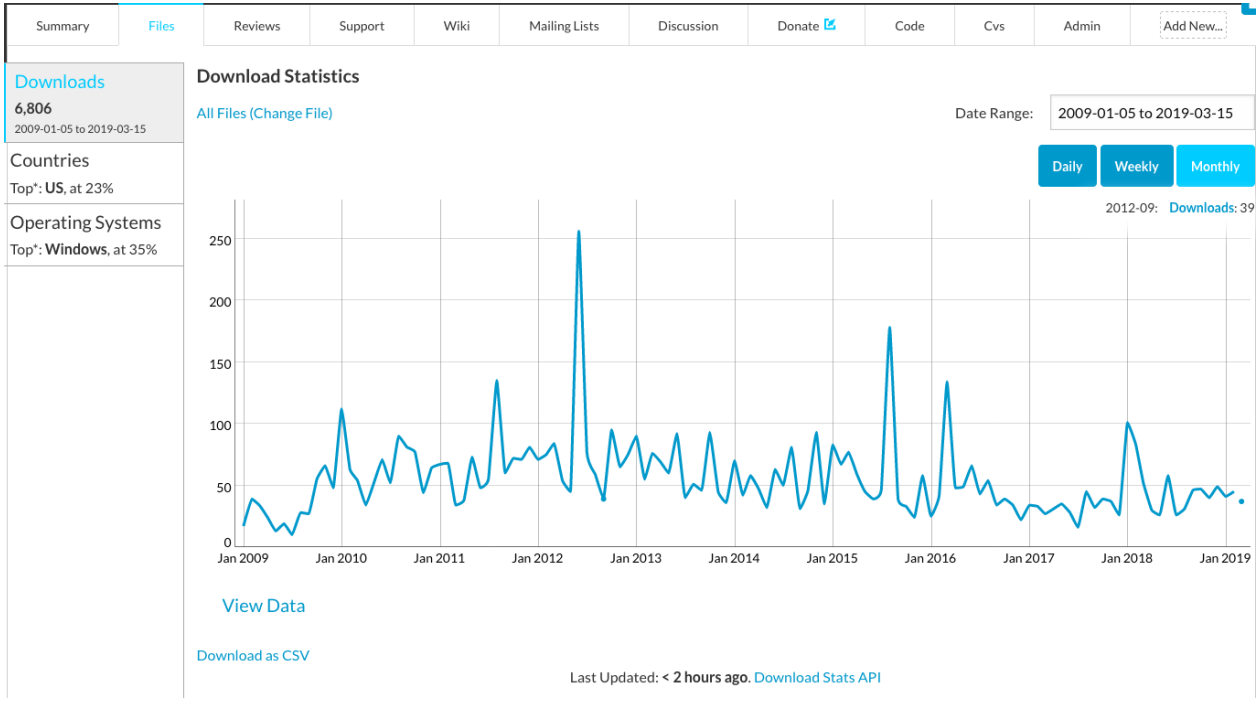
MLAPACK のコード例

Rsyev; 対称行列の対角化から抜粋

```
            Rlascl(uplo, 0, 0, One, sigma, n, n, A, lda, info);
        }
//Call DSYTRD to reduce symmetric matrix to tridiagonal form.
        inde = 1;
        indtau = inde + n;
        indwrk = indtau + n;
        llwork = *lwork - indwrk + 1;
        Rsytrd(uplo, n, &A[0], lda, &w[0], &work[inde - 1], &work[indtau -
            &work[indwrk - 1], llwork, &iinfo);
//For eigenvalues only, call DSTERF.  For eigenvectors, first call
//DORGTR to generate the orthogonal matrix, then call DSTEQR.
        if (!wantz) {
            Rsterf(n, &w[0], &work[inde - 1], info);
        } else {
            Rorgtr(uplo, n, A, lda, &work[indtau - 1], &work[indwrk - 1],
                &iinfo);
            Rsteqr(jobz, n, w, &work[inde - 1], A, lda, &work[indtau - 1],
        }
//If matrix was scaled, then rescale eigenvalues appropriately.
        if (iscale == 1) {
            if (*info == 0) {
```

MPLAPACK に関する事実

- Google 検索で ”Multiple precision BLAS” とするとほぼ MPLAPACK についてのみ
- ダウンロード 6806 回以上 (2019/3/15)



MBLAS の品質保証

BLAS は代数処理のみ

- Netlib BLAS 版を正しいとし、MPFR 版の正当化を行う。
- 色々な場合について値を MPFR 版 MBLAS, Netlib 版に代入して比較、差が小さいなら ok
- MPFR 版 MBLAS が正当化されたら、GMP 版, DD 版... と MPFR 版を比較する。
- BLAS は基本代数演算なので、この程度で ok

```
for (int k = MIN_K; k < MAX_K; k++) {
 for (int n = MIN_N; n < MAX_N; n++) {
  for (int m = MIN_M; m < MAX_M; m++) {
...
   for (int lda = minlda; lda < MAX_LDA; lda++) {
    for (int ldb = minldb; ldb < MAX_LDB; ldb++) {
     for (int ldc = max(1, m); ldc < MAX_LDC; ldc++) {

      Rgemm(transa, transb, m, n, k, alpha, A, lda, B, ldb, beta, C
      dgemm_f77(transa, transb, &m, &n, &k, &alphad, Ad, &lda,
                                            Bd, &ldb, &betad, Cd, &
...
```

MLAPACK の品質保証

収束の概念が入るのでとても難しくなる

-
- 収束の概念が LAPACK から入っている
  - 本質的に変わるが、結局 MBLAS のときと同じようなデバッグ方法で大体取れる
- 研究に使うとバグが発見されることがある (脇ら *et al*: seg fault, 無限ループなど。)

Raxpy のパフォーマンス

on Intel Core i7 920 (2.6GHz) / Ubuntu 10.04 / gcc 4.4.3

$$y \leftarrow \alpha x + y$$

`Raxpy` Flops 単位で. 括弧内は OpenMP による加速

MP Library(sign. digs.)	Flops (OpenMP)
DD(32)	130(570)M
QD(64)	13.7(67)M
GMP(77)	11.3(45)M
GMP(154)	7.6(32)M
MPFR(154)	3.7(17)M
GotoBLAS(16)	1.5G

Performance of Rgemv

on Intel Core i7 920 (2.6GHz) / Ubuntu 10.04 / gcc 4.4.3

$$y \leftarrow \alpha Ax + \beta y$$

Rgemv Flops 単位。

MP Library(sign. digs.)	Flops (OpenMP)
DD(32)	140M
QD(64)	13M
GMP(77)	11.1M
MPFR(77)	4.7M
GMP(154)	7.1M
MPFR(154)	3.7M
GotoBLAS(16)	3.8G

Performance of Rgemm

on Intel Core i7 920 (2.6GHz) / Ubuntu 10.04 / gcc 4.4.3

`Rgemm` Flops 単位。括弧内は OpenMP での高速化

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

MP Library(sign. digs.)	Flops (OpenMP)
DD(32)	136 (605)M
QD(64)	13.9 (63)M
GMP(77)	11.5 (44)M
MPFR(77)	4.6 (20)M
GMP(154)	7.2 (28) M
MPFR(154)	3.7 (16) M
GotoBLAS(16)	42.5G

Performance of Rsyev

on Intel Core i7 920 (2.6GHz) / Ubuntu 10.04 / gcc 4.4.3

`Rsyev` (300x300 対称行列、固有値固有ベクトルを求めるのにかかる時間)

$$AX = diag[\lambda_1, \lambda_2, \cdots \lambda_N]X$$

MP Library(sign. digs.)	seconds
DD(32)	2.4
QD(64)	25.6
GMP(77)	36.9
MPFR(77)	78.9
GMP(154)	64.0
MPFR(154)	111
GotoBLAS(16)	0.1

コンパイル on Ubuntu 16.04.6 LTS (GNU/Linux 4.4.0-137-generic x86_64)

ソースのみ。フルビルドを行う。ものすごく時間がかかる!

```
$ wget https://sourceforge.net/projects/mplapack/files/mpack/mpack%200.8.0/mpack-0.8.0.tar.gz/
$ tar xvfz download
$ cd mpack-0.8.0
$ ./configure --prefix=$HOME/mplapack/ --enable-gmp=yes --enable\
-debug=yes --enable-mpfr=yes --enable-qd=yes --enable-double=yes \
--enable-__float128=yes
$ cat /proc/cpuinfo  | grep Xeon | head -1
model name      : Intel(R) Xeon(R) CPU E5-2680 0 @ 2.70GHz
$ /usr/bin/time make -j16 2>&1 |tee log
...
13745.84user 1171.56system 38:21.68elapsed 648%CPU (0avgtext+0avgdata 238716maxresident)k
184inputs+41900152outputs (0major+562385275minor)pagefaults 0swaps
$ make install
```

MBLAS サンプル on Ubuntu 16.04.6 LTS (GNU/Linux 4.4.0-137-generic x86_64)

MPFR 版の Rgemm(=dgemm の多倍長精度版) 行列行列積を行っている。Octave で検算もできる。

```
$ cd mpack-0.8.0/examples/mblas
$ ./Rgemm.mpfr
Rgemm demo...
A =[ [ 1.00e+00, 8.00e+00, 3.00e+00]; [ 0.00e+00, 1.00e+01, 8.00e+00]; [ 9.00e+00, -5.00e+00,
B =[ [ 9.00e+00, 8.00e+00, 3.00e+00]; [ 3.00e+00, -1.10e+01, 0.00e+00]; [ -8.00e+00, 6.00e+00
C =[ [ 3.00e+00, 3.00e+00, 0.00e+00]; [ 8.00e+00, 4.00e+00, 8.00e+00]; [ 6.00e+00, 1.00e+00,
alpha = 3.000e+00
beta  = -2.000e+00
ans =[ [ 2.10e+01, -1.92e+02, 1.80e+01]; [ -1.18e+02, -1.94e+02, 8.00e+00]; [ 2.10e+02, 3.61e
#please check by Matlab or Octave following and ans above
alpha * A * B + beta * C =
```

MBLAS サンプル on Ubuntu 16.04.6 LTS (GNU/Linux 4.4.0-137-generic x86_64)

DD 版の Rgemm(=dgemm の多倍長精度版) 行列行列積を行っている。先ほどとは DD に変えただけ。この例だと精度をみているわけでないので全く変化しない。

```
$ cd mpack-0.8.0/examples/mblas
$ ./Rgemm.dd
Rgemm demo...
A =[ [ 1.00e+00, 8.00e+00, 3.00e+00]; [ 2.50e+00, 1.00e+01, 8.00e+00]; [ 9.00e+00, -5.00e+00,
B =[ [ 9.00e+00, 8.00e+00, 3.00e+00]; [ 3.00e+00, -1.10e+01, 4.80e+00]; [ -8.00e+00, 6.00e+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,
alpha = 3.000e+00
beta  = -2.000e+00
ans =[ [ 2.10e+01, -1.92e+02, 1.31e+02]; [ -5.05e+01, -1.34e+02, 1.74e+02]; [ 2.10e+02, 3.61e
#please check by Matlab or Octave following and ans above
alpha * A * B + beta * C =
```

テスト:ヒルベルト行列

ヒルベルト行列 H は $n \times n$ 行列で, 各要素 H_{ij} が

$$H_{ij} = \frac{1}{i+j-1} \quad (1 \le i, j \le n)$$

となるものである。ヒルベルト行列は n が大きくなると条件数が指数関数的に大きくなる: $n = 12$ のとき 1.7132×10^{16} 倍精度では逆行列が求まらない。example は MPFR で $n = 1 \cdots 50$ でヒルベルト行列を作り

$$I - HH^{-1}$$

誤差を求める、ということをする。精度は 1024bit=10 進数 308 桁としている。

```
$ cd mpack-0.8.0/examples/mlapack
$ ./hilbert_mpfr | less
# inversion of Hilbert matrix of order n=1
A = [ [ 1.00e+00] ]
invA = [ [ 1.00e+00] ]
A * invA =[ [ 1.00e+00] ]
Infnorm(A*invA)=0.0000000000000000e+00
# inversion of Hilbert matrix of order n=2
A = [ [ 1.00e+00, 5.00e-01]; [ 5.00e-01, 3.33e-01] ]
invA = [ [ 4.00e+00, -6.00e+00]; [ -6.00e+00, 1.20e+01] ]
A * invA =[ [ 1.00e+00, 0.00e+00]; [ 0.00e+00, 1.00e+00] ]
Infnorm(A*invA)=0.0000000000000000e+00
# inversion of Hilbert matrix of order n=3
A = [ [ 1.00e+00, 5.00e-01, 3.33e-01]; [ 5.00e-01, 3.33e-01, 2.50e-01]; [ 3.33e-01, 2.50e-01,
invA = [ [ 9.00e+00, -3.60e+01, 3.00e+01]; [ -3.60e+01, 1.92e+02, -1.80e+02]; [ 3.00e+01, -1.8
A * invA =[ [ 1.00e+00, 9.55e-153, 0.00e+00]; [ -2.98e-153, 1.00e+00, 0.00e+00]; [ -1.79e-153,
Infnorm(A*invA)=9.5466761359362646e-153
# inversion of Hilbert matrix of order n=4
```

テスト:ヒルベルト行列:精度を実行時に指定

ヒルベルト行列 H は $n \times n$ 行列で, 各要素 H_{ij} が

$$H_{ij} = \frac{1}{i+j-1} \quad (1 \le i, j \le n)$$

となるものである。ヒルベルト行列は n が大きくなると条件数が指数関数的に大きくなる: $n = 12$ のとき 1.7132×10^{16} 倍精度では逆行列が求まらない。example は MPFR で $n = 1 \cdots 50$ でヒルベルト行列を作り

$$I - HH^{-1}$$

誤差を求める、ということをする。デフォルトで精度は 512bit=10 進数 154 桁。精度を実行時に指定する。$n = 50$ の結果として精度を 512bit/1024bit/2048bit で求めてみた。

```
$ cd mpack-0.8.0/examples/mlapack
$ ./hilbert_mpfr
...
Infnorm(A*invA)=1.9467073866867010e-75
$ MPACK_MPFR_PRECISION=1024 ./hilbert_mpfr
...
Infnorm(A*invA)=9.8379839296799375e-229
$ MPACK_MPFR_PRECISION=2048 ./hilbert_mpfr
...
Infnorm(A*invA)=0.0000000000000000e+00
```

MPLAPACK へのコントリビューション

- MPLAPACK へのコードなどのコントリビューションは大歓迎。今までパッチ提供、バグ報告などは数件あった。
- このルーチンが欲しいという問い合わせ、要望は多数くるが、なかなか応えられてない...
- 2 条項 BSD ライセンスでコードを提供していただけるとありがたい。
- 開発者権限も発行できるといいな...

MPLAPACK 0.8.0: 高精度版の BLAS and LAPACK

http://mplapack.sourceforge.net/

中田真秀 @ 理研

- MPLAPACK: multiple precision version of BLAS and LAPACK.
- ビルディングブロック、参照実装、API の提供
- Version 0.8.0 (2012/12/25); 状況: MBLAS 完成, and 100 MLAPACK ルーチン.
- double-double 精度の Rgemm 作成, リリース間近; 他の研究 [椋木, 高橋 2010, 2011], [中里 2010])