第237回お試しアカウント付き講習会
2024年12月13日

「任意精度・精度保証ライブラリ」

本講習会の内容

1. 注意事項
2. 本講習会での質問の仕方、ZoomおよびSlackの使い方
3. 任意精度
4. 精度保証
5. 実習

講習会の注意事項

- 1か月利用可能なアカウントが配布されます。
  - 利用規定に記載の用途以外に使用しないでください。
    - ✓ 研究、教育、社会貢献のために使用
  - 1か月経過後にはアカウントが削除されるため、ファイルも削除されます。
  - アカウントの継続利用のためには、講習会を最後まで受講いただく必要があります。
- 企業の方がスパコンの利用申請をする場合、センターが開催するいずれかの講習会への参加が要件となります。
  - 講習会の最後までの受講が申し込みの要件となります。
- 本講習会に関する質問は
  kawai@cc.nagoya-u.ac.jp
  に送付ください。センターのスパコン相談窓口等には送付しないようご注意ください。

質問の方法

任意のタイミングで受け付けます。

- 講義中の質問方法
  - Zoomの挙手後の発言
  - Slackへの書き込み

- 実習中の質問方法
  - Zoomの挙手後の発言
  - Slackへの書き込み
  - Zoomの“ヘルプを求める”機能の利用
    - ✓ 画面を共有してのヘルプ

本講習会の目的

任意精度、精度保証ライブラリの利用方法のご紹介
対応言語はC、Fortran

- 任意精度
  - 必要な精度を確保と、低精度化による計算時間短縮を実現
  - 実装が手間→ライブラリを利用
  - 低精度化のノウハウ
    - ✓ 高速化
    - ✓ コンパイル環境の整え方（マクロの使い方）
- 精度保証
  - 低精度化はいいが、結果は妥当なのか？
  - ライブラリがない
    - ✓ 反復法限定ではあるが、作成したので、ご紹介

本講習会の内容

全3時間

- 任意精度：1時間
- 任意精度ライブラリの演習：45分
- 精度保証：1時間
- 精度保証ライブラリの演習：15分

任意精度

IEEE754で規格化”されていない”型

- IEEE754で規格化されている型
  - FP16（半精度）
  - FP32（単精度）
  - FP64（倍精度）
  - FP128（4倍精度）
- 任意精度
  - 上記の型から可数部、指数部を任意に変更
  - 多倍長 (倍精度より高精度側)
    - ✓ 高精度演算に利用
    - ✓ 演算方法を変更
    - ✓ ライブラリの種類は多
  - 低精度（倍精度より低精度側）
    - ✓ 今回の講習のターゲット
    - ✓ メモリバンド幅律速なアプリの高速化
    - ✓ メモリ使用量削減用途？
    - ✓ 既存の演算機を使用、メモリへの格納時に圧縮 (Memory Accessor)
    - ✓ ライブラリの種類は少

Memory Accessor

任意精度（低）で導入される概念

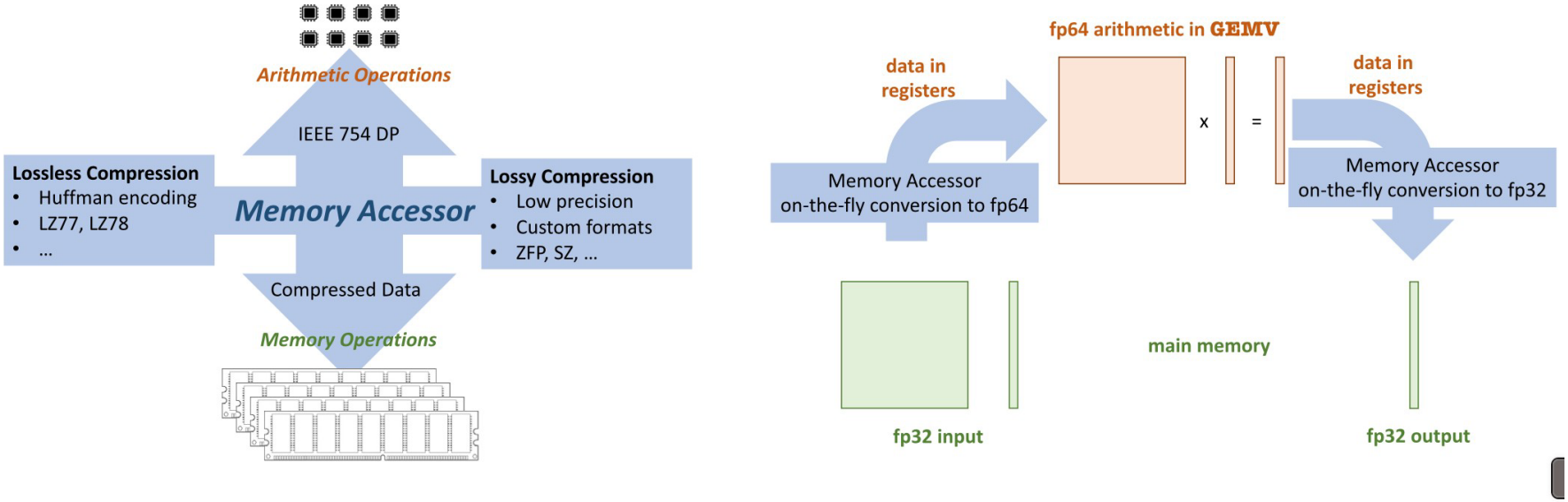
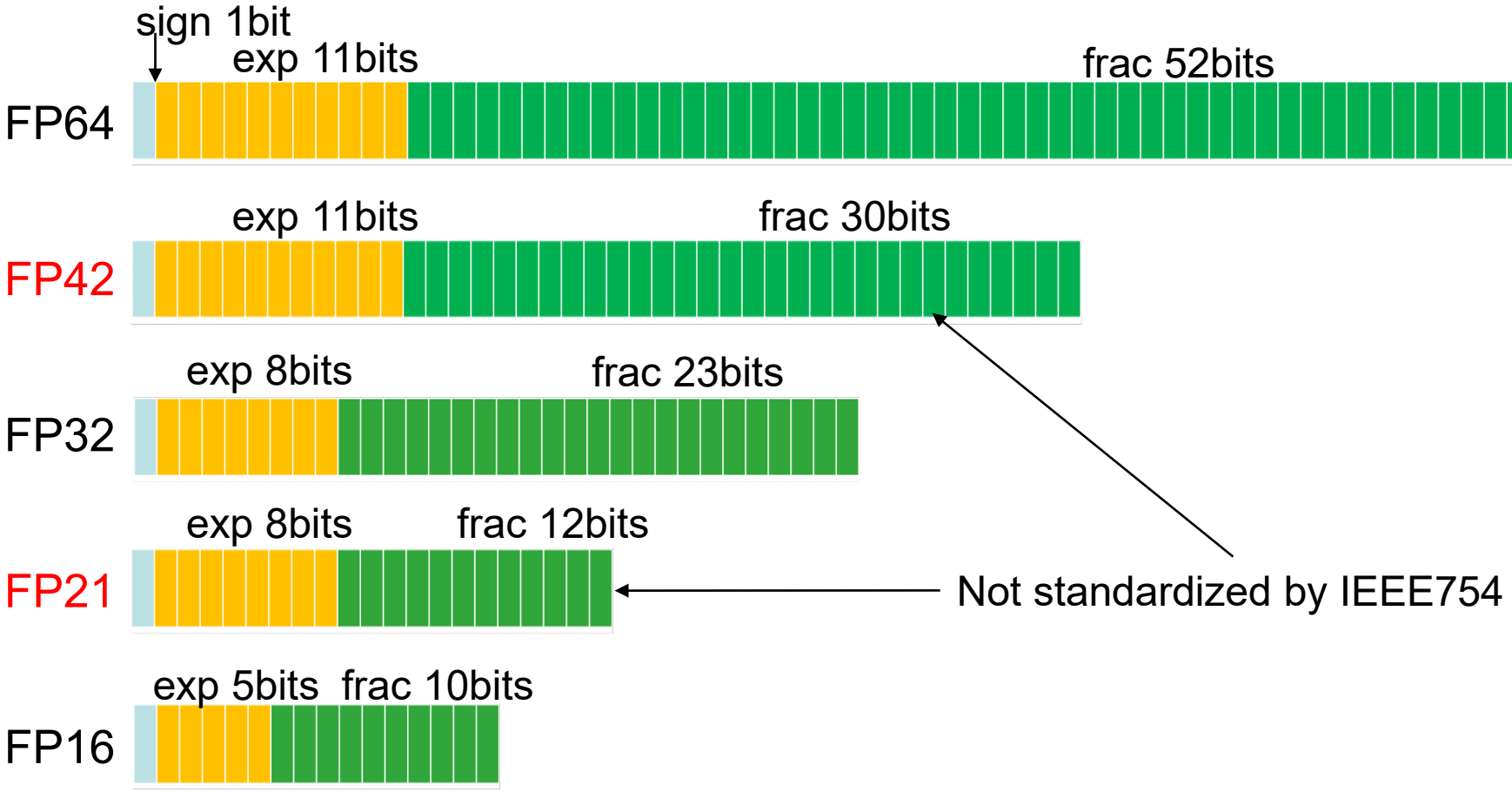


FIGURE 2 Overall idea of the memory accessor compressing data on-the-fly in memory access (left), and an example of use from a matrix-vector product (GEMV) kernel using `fp32` as memory format and `fp64` as arithmetic format (right) 注1

- メモリ(配列)への格納時に圧縮
  - Lossless
  - Lossy
    - ✓ AoS : dafpp (FP21, FP42)
    - ✓ SoA : RpFp (FP56, FP48, FP40, FP24)
- 演算時には、より高精度なIEEE754で規格化された型にキャスト

注1 : D. Mukunoki, et.al. "Sparse Matrix-Vector Multiplication with Reduced-Precision Memory Accessor," 2023 IEEE 16th International Symposium on Embedded Multicore/Many-core Systems-on-Chip (MCSoC), Singapore, 2023, pp. 608-615

FP21、FP42



- 可数部を落として低精度化
- データ転送量はFP21はFP32に対して、FP42はFP64に対してそれぞれ2/3に
- 演算量は減らない

FP21とFP42の表現能力

FP21とFP42の10進数に換算した場合の表現能力は以下の通り

10進数に換算した場合の表現応力

データ型	仮数部：表現可能桁数	指数部：最大冪指数
FP64	15.95	308
FP42	9.33	308
FP32	7.22	38
FP21	3.91	38
FP16	3.31	5

仮数部の表現桁数は10進数の桁数をy、仮数部のビット数をxとして、
以下のように算出

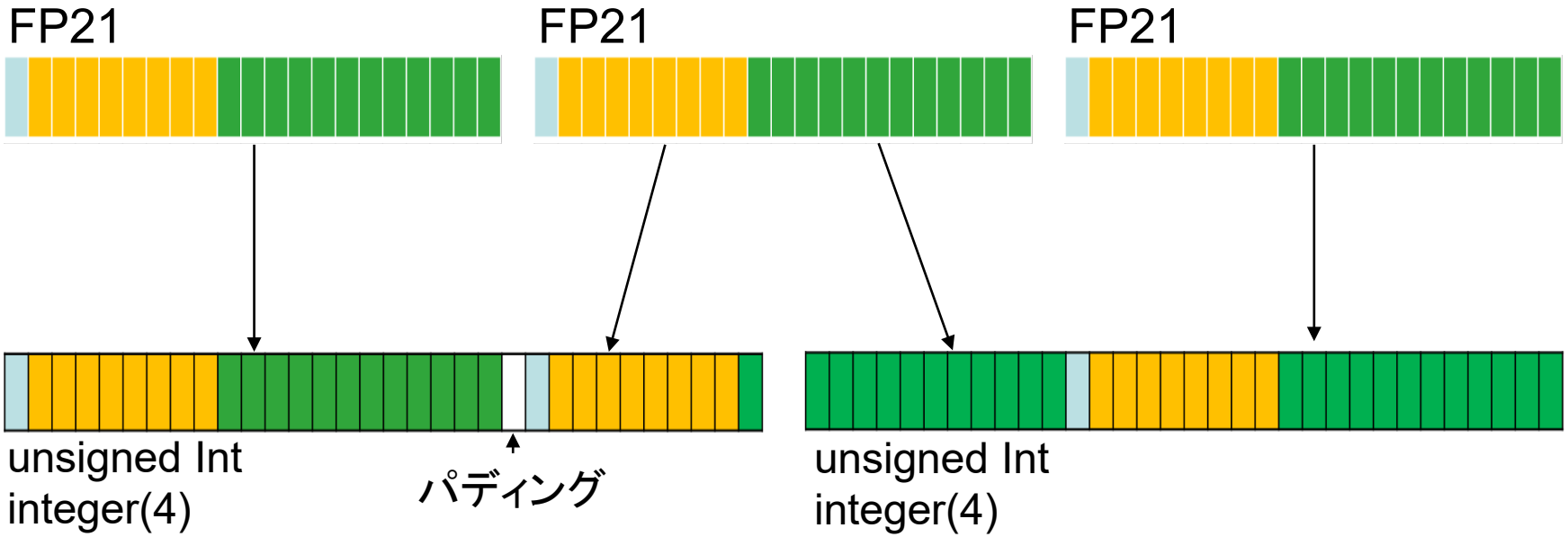
$$10^y = 2^{x+1}$$

x+1はhidden bitによるもの

$$y = (x+1)\log_{10} 2$$

FP21の圧縮方法

3つのFP21を2つのinteger(32-bit)に格納



- Packingとも呼称
- 3つのFP21を1つの単位として扱う
- 何故long int, integer(8)じゃないのかは後述
- FP42は2つのunsigned long, integer(8)に格納

FP32 → FP21 (C言語)

FP32→FP21

```
#define fp21x3 unsigned int

inline void floatx3_to_fp21x3(float const a1,
 float const a2, float const a3, fp21x3 *b)
{
  fp21x3 c;

  b[0]  = ((*(int *)&a1) & 0xfffff800) >>11;

  c = ((*(int *)&a2) & 0xfffff800);
  b[0] |= c << 10;
  b[1]  = c >> 22;

  b[1] |= ((*(int *)&a3) & 0xfffff800);
}
```

FP21型のデータ3つを32bit整数型2つで格納.

- 基本的な部分は論理演算、シフト演算で実装可能
- bit情報を変えないデータ型変更をどうするか
  → ポインタで実現
  (C++のreinterpret_cast)
- 低オーバーヘッドかつ効率的なSIMD化のために、コンパイラによるインライン展開が必須

2つのunsigned intを使う理由

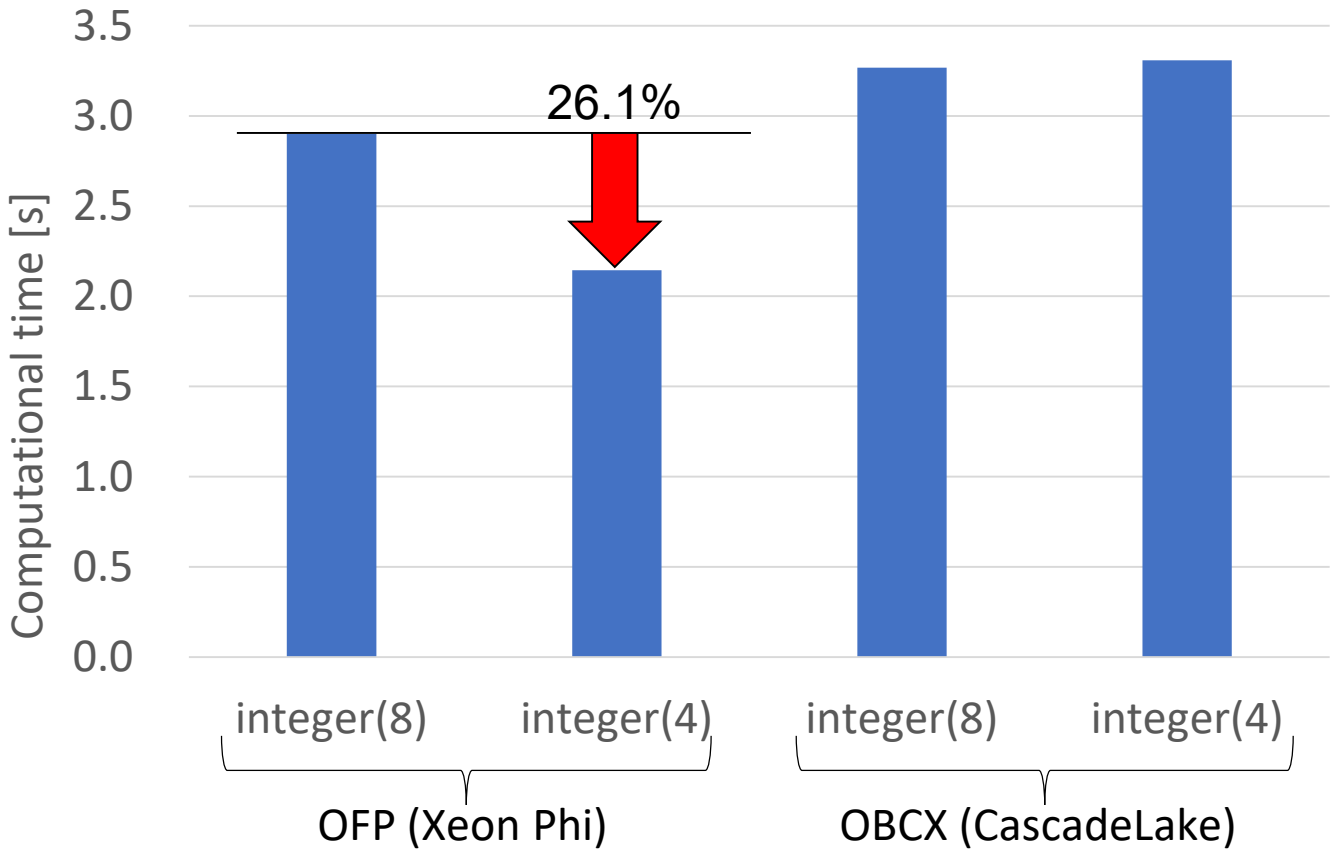
- 1SIMD命令で計算する要素数はデータの型で決定
  512bit SIMDだとすると、
  - FP64 → 8要素
  - FP32 → 16要素
- 正確にはSIMD化するループ内で扱われる最もbit長の長いデータ型で決定？
  - ほとんどがFP32とInteger、1変数だけFP64 → 8要素
  - ほとんどがFP32とInteger、1変数だけlong int → 8要素

Floatの3要素をunsigned long int(64bit)1要素に格納 → 8要素
希望は16要素

unsigned longとunsigned intの性能差

システムによっては効果あり

メモリバンド幅に対してフロントエンドや演算性能の弱いシステムでは必要は対処



インライン展開

小規模な関数の呼び出しを呼び出し元に展開

ソフトウェア最適化の1つ

- メリット
  - 命令の実行順序の最適化（パイプライニング）
  - SIMD化の促進
- デメリット
  - 命令キャッシュのヒット率低下
  - バイナリの肥大化

関数のインライン展開を促進・確認する方法

コンパイラ	促進オプション	確認オプション	結果例
Intel	-flto	-qopt-report (ファイルに出力)	-> INLINE: floatx3_to_fp21x3_f_ ld-temp.o (37,8)
gcc, gfortran	-flto	-fopt-info-inline	optimized: Inlining cast_fp21x3_to_fp32_/11 into fp21x3_to_floatx3_f_/12.

IntelコンパイラはOneAPIベース

- C : icx、Fortran : ifx

Parallel studioのオプションは以下

- C : icc -ipo, Fortran : ifort -ipo

FP32 → FP21 (Fortran)

FP32→FP21

```
#define fp21x3 integer(4)

function fp32x3_to_fp21x3_f(a1, a2, a3) result(b)
  implicit none
  real(4), intent(in) :: a1, a2, a3
  fp21x3 :: b(2)
  fp21x3 c
  call cast_fp32_to_fp21x3(a1, c)
  b(1) = shiftr(iand(c, int(Z'fffff800', 4)), 11)
  call cast_fp32_to_fp21x3(a2, c)
  c = iand(c, int(Z'fffff800', 4))
  b(1) = ior(b(1), shiftl(c, 10))
  b(2) = shiftr(c, 22)
  call cast_fp32_to_fp21x3(a3, c)
  b(2) = ior(b(2), iand(c, int(Z'fffff800', 4)))
end function fp32x3_to_fp21x3_f

subroutine cast_fp32_to_fp21x3(a, b)
  implicit none
  fp21x3, intent(in)  :: a
  fp21x3, intent(out) :: b
  b = a
end subroutine cast_fp32_to_fp21x3
```

ここの関数呼び出しで型キャスト

- 基本的な部分はCと同じ
- bit情報を変えないデータ型変更がFortranでは困難
  →サブルーチンへの引数をごまかす形で実装
  - ✓ 仕様上、引数の型の一致を保証するのはユーザー
  - ✓ ちょっとグレー

- Intel fortran, gfortranでインライン展開を確認 (富士通コンパイラは×)
  - ✓ gfortranでは、専用のオプション(-fallow-argument-mismatch)が必須
  - ✓ 引数の型が不一致なコードが世の中にあるのは分かっているが、それはnon-conformingだ。しゃーねーからこのオプションでErrorからWarningに落としてやるよ。(とman gfortranに書かれている。(意訳込み))

任意精度ライブラリの関数一覧 (C)

- void fp21x3_to_floatx3(fp21x3 *a, float *b1, float *b2, float *b3);
  FP21からfloat(3要素)にキャスト
- void fp21x3_to_float(fp21x3 *a, float *b, int const pos);
  FP21からfloat(1要素)にキャスト
- void floatx3_to_fp21x3(float const a1, float const a2, float const a3, fp21x3 *b);
  float(3要素)からFP21にキャスト
- void float_to_fp21x3(float const a, fp21x3 *b, int const pos);
  float(1要素)からFP21にキャスト
- void init_fp21x3_zero(fp21x3 *a);
  FP21の0初期化

- void fp42x3_to_doublex3(fp42x3 *a, double *b1, double *b2, double *b3);
  FP42からdouble(3要素)にキャスト
- void fp42x3_to_double(fp42x3 *a, double *b, int const pos);
  FP42からdouble(1要素)にキャスト
- void doublex3_to_fp42x3(double const a1, double const a2, double const a3, fp42x3 *b);
  double(3要素)からFP42にキャスト
- void double_to_fp42x3(double const a, fp42x3 *b, int const pos);
  double(1要素)からFP42にキャスト
- void init_fp42x3_zero(fp42x3 *a);
  FP42の0初期化

任意精度ライブラリの関数一覧 (Fortran) 1/2

```
subroutine fp21x3_to_floatx3_f(a, b1, b2, b3)
  fp21x3, intent(in) :: a(2)
  real(4), intent(out) :: b1, b2, b3
end subroutine fp21x3_to_floatx3_f

function fp21x3_to_float_f(a, pos) result(b)
  fp21x3,  intent(in) :: a(2)
  integer, intent(in) :: pos
  real(4) b
end function fp21x3_to_float_f

subroutine floatx3_to_fp21x3_f(a1, a2, a3, b)
  real(4), intent(in)  :: a1, a2, a3
  fp21x3,  intent(out) :: b(2)
end subroutine floatx3_to_fp21x3_f

function floatx3_to_fp21x3_f(a, pos) result(b)
  real(4), intent(in) :: a
  integer, intent(in) :: pos
  fp21x3 :: b(2)
end function floatx3_to_fp21x3_f

function init_fp21x3_zero_f() result(b)
  fp21x3 :: b(2)
end function init_fp21x3_zero_f
```

- インターフェイスは基本Cと同じ
- 3要素ずつ扱う関数はsubroutine
- 1要素を扱う関数と初期化関数はfunction

任意精度ライブラリの関数一覧 (Fortran) 2/2

```
subroutine fp42x3_to_doublex3_f(a, b1, b2, b3)
  fp42x3, intent(in) :: a(2)
  real(8), intent(out) :: b1, b2, b3
end subroutine fp42x3_to_doublex3_f

function fp42x3_to_double_f(a, pos), result(b)
  fp42x3,  intent(in) :: a(2)
  integer, intent(in) :: pos
  real(8) b
end function fp42x3_to_double_f

subroutine doublex3_to_fp42x3_f(a1, a2, a3, b)
  real(8), intent(in)  :: a1, a2, a3
  fp42x3,  intent(out) :: b(2)
end subroutine doublex3_to_fp42x3_f

function double_to_fp42x3_f(a, pos), result(b)
  real(8), intent(in) :: a
  integer, intent(in) :: pos
  fp42x3 :: b(2)
end function double_to_fp42x3_f

function init_fp42x3_zero_f() result(b)
  fp42x3 :: b(2)
end function init_fp42x3_zero_f
```

- FP42版はFP21と基本同
- 引数、返値が違うぐらい。

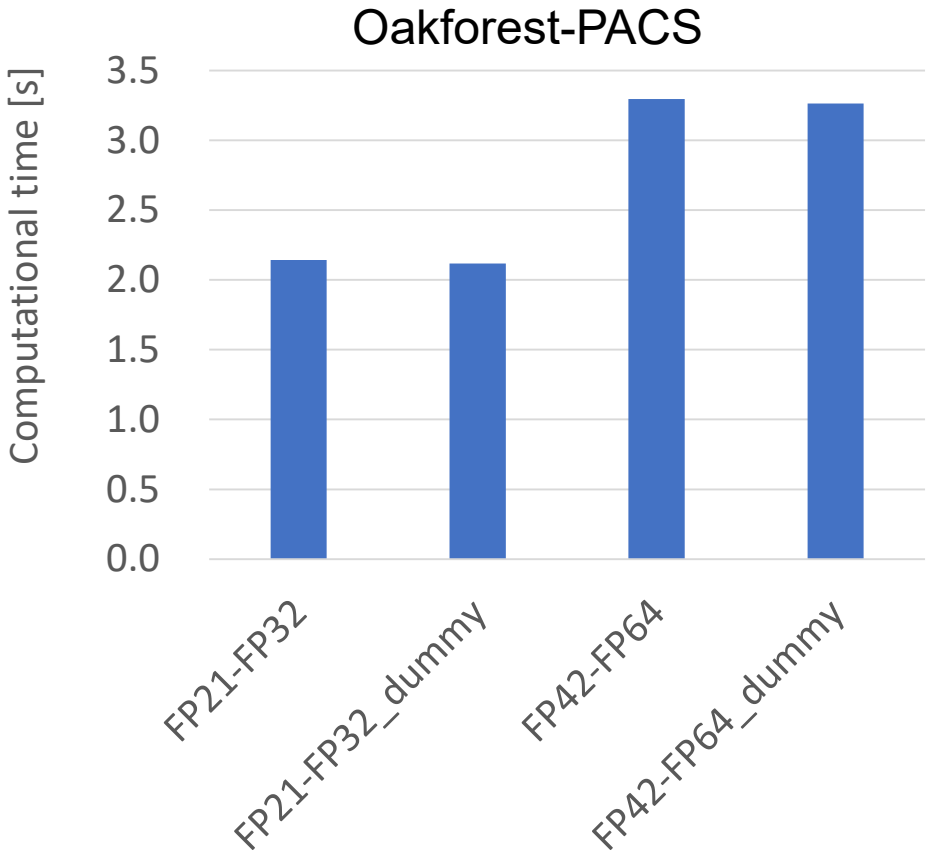
任意精度ライブラリの使い方

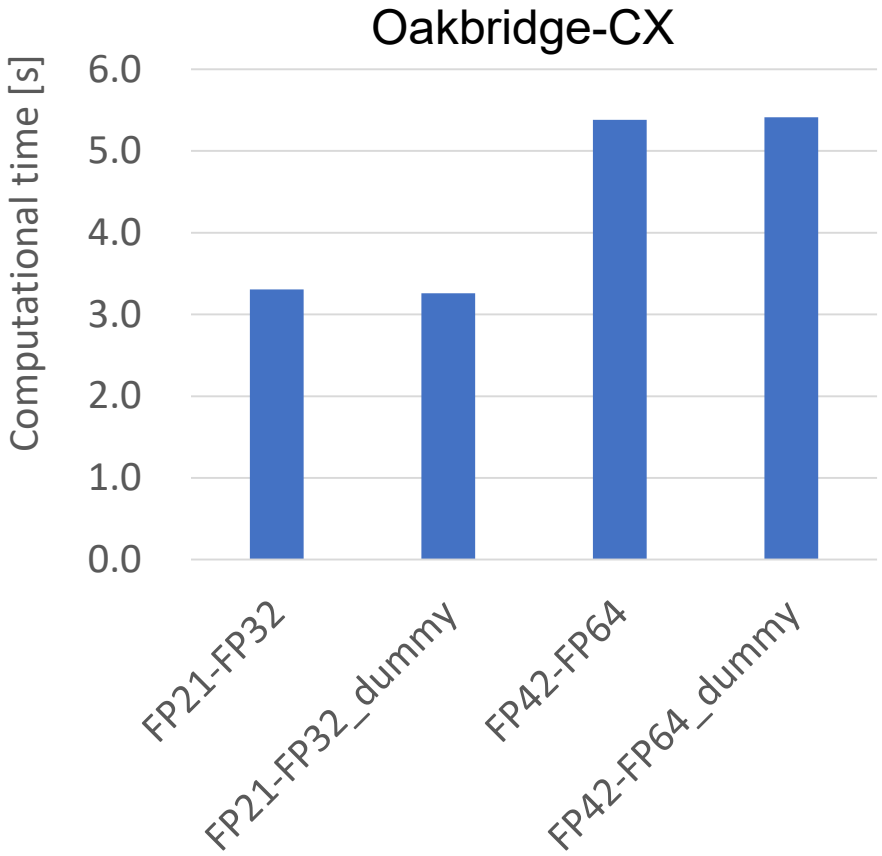
```c
#include "./libdafpp_fix.h"

int main(void){
  float fx, fy, fz;
  float fxt, fyt, fzt;
  fp21x3 ap21[2];

  //Type casting from FP32 to FP21
  floatx3_to_fp21x3(fx, fy, fz, ap21);

  //Type casting from FP21 to FP32
  fp21x3_to_floatx3(ap21, &fxt, &fyt, &fzt);

}
```

```fortran
#include "libdafpp_fix_f.h"

program main
  use dafpp
  implicit none
  real(4) fx, fy, fz
  real(4) fxt, fyt, fzt
  fp21x3 :: ap21(2)

  !Type casting from FP32 to FP21
  call floatx3_to_fp21x3_f(fx, fy, fz, ap21)

  !Type casting from FP21 to FP32
  call fp21x3_to_floatx3_f(ap21, fxt, fyt, fzt)

end program
```

型変換のオーバーヘッド

型キャストによるオーバーヘッドが小さいのは確認済





dummy ：右のように書き換え
→ 計算結果がおかしくなるが、
同じアクセス量で比較

fp21x3_to_floatx3(array, f1, f2, f3)
→ f1 = array(1)
f2 = array(1)
f3 = array(2)

実行結果

FP21の実行結果一例

NaNやInfの場合も含めて正しく型キャストが行われていることを確認

FP32↔FP21 (FP32 = FP21の形で表記)

```
Check conv, 1.0000000000000000 = 1.0000000000000000,
           -1.0000000000000000 = -1.0000000000000000,
            0.0153462402522564 =  0.0153446197509766

Check conv,       Infinity =       Infinity,      -Infinity =      -Infinity,          NaN =          NaN

Check conv, 1.0000001192092896 = 1.0000000000000000,
            1.0001220703125000 =  1.0000000000000000,
            1.0003662109375000 = 1.0002441406250000
```

注意点

性能向上を実現するためには

- アプリケーションがチューニングされており、メモリバンド幅律速となっている。
- 3要素を同時に扱って初めて性能向上が期待できる。
- インライン展開が必須
  - ✓ インライン展開が実施されたか要確認
  - ✓ Cはinline指定子がある。
  - ✓ Fortranはコンパイラに期待するしかない。

単一コードでの複数データ型の扱い方 (header)

基本的にはマクロで

Sample C header (precision.h)

```
#include "libdafpp_fix.h"

#ifdef _fp42
#include "libdafpp_fix.h"
#define flag_ap
#define val fp42x3
#define temp_val double
#define load_ap fp42x3_to_doublex3
#define store_ap doublex3_to_fp42x3
#elif _single
#define val float
#define temp_val float
#elif _fp21
#include "libdafpp_fix.h"
#define flag_ap
#define val fp21x3
#define temp_val double
#define load_ap fp21x3_to_floatx3
#define store_ap floatx3_to_fp21x3
#else
#define val double
#define temp_val double
#endif
```

Sample Fortran include file (precision.inc)

```
#include "libdafpp_fix_f.h"

#ifdef _fp42
#include "libdafpp_fix_f.h"
#define flag_ap
#define val fp42x3
#define temp_val real(8)
#define load_ap fp42x3_to_doublex3_f
#define store_ap doublex3_to_fp42x3_f
#elif _single
#define val real(4)
#define temp_val real(4)
#elif _fp21
#include "libdafpp_fix_f.h"
#define flag_ap
#define val fp21x3
#define temp_val real(8)
#define load_ap fp21x3_to_floatx3_f
#define store_ap floatx3_to_fp21x3_f
#else
#define val real(8)
#define temp_val real(8)
#endif
```

- マクロで型を変更
  - -D_fp21
  - -D_single
  - -D_fp21
  - -D_double

  マクロ指定なし→double
- FP21かFP42かで、配列への格納、取り出し関数名も変わるため、これもdefineで切り替え
- 任意精度か否かはflag_apで切り替え
- temp_valは一時変数

プリプロセッサの使用方法

マクロを処理するのはプリプロセッサ

Cはオプションも不要で、何も考える必要なし
FortranでCプリプロセッサを使う方法

- オプションの付与
- 拡張子の頭文字を大文字に
  - ✓ .F、.F90など

	オプション	確認オプション
Intel fortran	-fpp	-dM -E
gfortran	-cpp	-dM -E

単一コードでの複数データ型の扱い方 (src)

宣言したマクロを使って、精度毎の違いを吸収

```
#include "precision.h"

void test (val *array, int const N){
  temp_val a1, a2, a3;

#ifdef flag_ap
  strd_ls = 2
#else
  strd_ls = 3
#endif

for(int i = 0; i < N; i+=strd_ls){
#ifdef flag_ap
  load_ap(array[i], a1, a2, a3);
#else
  a1 = array[i];
  a2 = array[i+1];
  a3 = array[i+2];
#endif
  }
}
```

```
#include "precision.inc"

subroutine test(array, N)
  val, intent(in) :: array(:)
  integer, intent(in) :: N
  temp_val a1, a2, a3

#ifdef flag_ap
  strd_ls = 2
#else
  strd_ls = 3
#endif

do i = 1, N, ls_strd
#ifdef flag_ap
  call load_ap(array(i), a1, a2, a3)
#else
  a1 = array(i)
  a2 = array(i+1)
  a3 = array(i+2)
#endif
end subroutine test
```

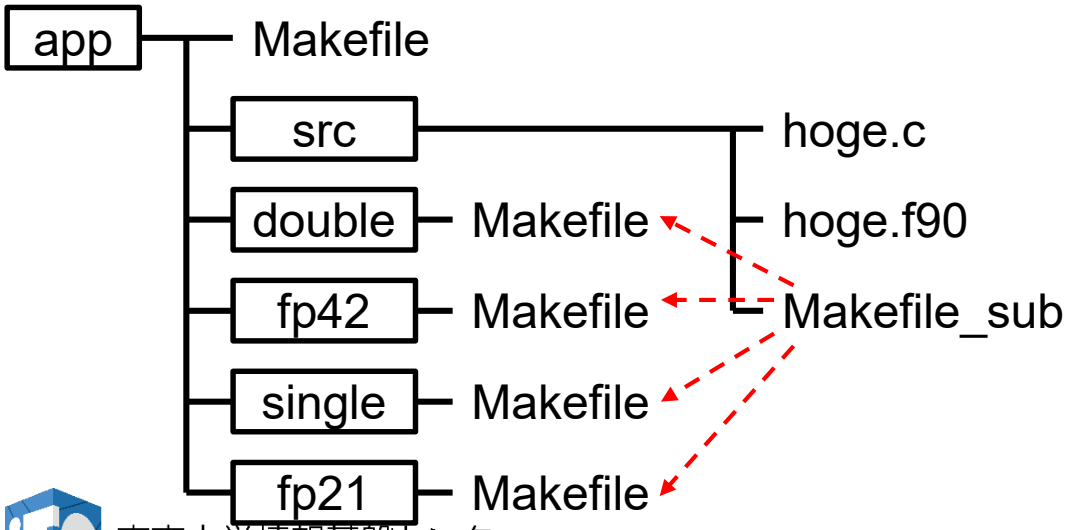
- 任意精度か否かで配列からtemp_valへのコピーを変更
- 配列アクセス時のストライド幅が変わるため、strd_lsで吸収

単一コードでの複数データ型の扱い方 (Makefile)

精度毎に異なるディレクトリに出力がおすすめ？

- 何も考えない（マクロだけ切り替えてコンパイル）
  - ✓ 精度を変えるごとに全部コンパイルし直し
- コンパイル時のオブジェクト名を変更
  - ✓ 全てのソースの依存関係を書く必要あり
  - ✓ suffix ruleは使いたい
    - .c.o : や .f90.o :

講師のやり方は、精度毎のディレクトリを作って
Makefileのシンボリックリンクを配置



Makefile サンプル

```
OBJs :=  hoge.o test.o
VPATH := $(DUR_DIR)/src
export

app_double :
        $(MAKE) -C double-double
TARGET=$(DIR_BIN)/$@
OBJ_SOLV=$(OBJ_SOLV) TYPE=double
```

Makefile_sub サンプル

```
.SUFFIXES:
.SUFFIXES: .f .f90 .c .o

$(TARGET) : $(OBJs)
        $(LINKER) -o $@ $^

.f90.o :
        $(FC) -c $(F90FLAGS) -D_$(TYPE) $<

.c.o   :
        $(CC) -c $(CFLAGS) -D_$(TYPE) $<
```

複数の型でコンパイルしたコードを同時に使う

マングリングのような仕組みが必要 → マクロで代用

- 型の異なる複数のパターンで単一のコードをコンパイル
  → 関数名が同じため、リンクエラーが発生
- C++だと関数名に引数の型などの情報を入れて関数名を決定
  →マングリング(名前修飾)

先の型を変更するマクロで関数名もコントロール

Cの関数名をマクロで変更するサンプル

```
#include 'precision.h'
#define CON(a) testsub_ ## a
#define SUFNAME(a) CON(a)

void SUFNAME(val)
```

-D_doubleの場合、
testsub_doubleとなる。
注：valはprecision.の中で
defineされたマクロ(3頁前)

Fortranの関数名をマクロで変更するサンプル

```
#include 'precision.inc'
#define PASTE(a) a
#define SUFNAME(a,b) PASTE(a)b

subroutine SUFNAME(testsub_,val)
```

Fortranでは何故かトークン
連結演算子(##)がうまく使え
なったので、左例のような書
き方で対処

参考コード https://bitbucket.org/naosou/p3d_dynamic_precision/src/master/

総称手続きを使う場合 (Fortran)

使う可能性がある関数は宣言
コンパイルしなかった型に関してはダミーモジュールを宣言

```
#if !defined(double)
module mod_type_double
  type st_type_double
     logical :: data_double = .false.
  end type st_type_double
end module mod_type_double
#endif
#if !defined(single)
module mod_type_single
  type st_type_single
     logical :: data_single = .false.
  end type st_type_single
end module mod_type_single
#endif

module mod_sample
  use mod_type_double
  use mod_type_single
```

```
  interface sample_sub
    module proceure test_double, test_single
  end interface

contains

  subroutine test_double()

  end subroutine test_double

  subroutine test_single()

  end subroutine test_single
end module mod_sample
```

“該当の型がコンパイルされていなければ、module procedureに含めない”も手

本講習会の内容

全3時間

- 任意精度：1時間
- **任意精度ライブラリの演習：45分**
- 精度保証：1時間
- 精度保証ライブラリの演習：15分

コンパイル、実行環境整備

Wisteriaにログイン後、以下を実行
$ module load intel/2024.0.0

以下はバッチファイルの一例
演習ではこちらを編集してご利用ください。

```
#!/bin/sh
#------ pjsub option --------#
#PJM -L rscgrp=tutorial-a
#PJM -g gt00
#PJM -L node=1
#PJM -L elapse=0:10:00
#PJM -j
#------- Program execution -------#
module load intel/2024.0.0
実行プログラム
```

コンパイルやジョブの投入は
/data/01/gt00/ユーザー名
ディレクトリ内で行ってください。

任意精度ライブラリ 演習 1/3

ライブラリに付属のテストコードのコンパイル、実行

```
$ git clone https://naosou@bitbucket.org/naosou/dynamic_precision.git
$ cd dynamic_precision
$ ls
libdafpp_fix.h          C言語用ヘッダー
dafpp_fix_prec_c.c      FP21, FP42ライブラリ (C言語版)
libdafpp_fix_f.h
dafpp_fix_prec_f.f90
test_c.c
test_f.f90
Makefile
README.md
```

C言語

```
$ make test_c
$ ./test_c
```

Fortran

```
$ make test_f
$ ./test_f
```

ここでのコード実行は非常に軽量のため、ログインノードでも問題ありません。
ただし、以降は重たくなるため、バッチファイルを生成の上、ジョブに投入してください。

任意精度ライブラリ 演習 2/3

CRS形式の行列ベクトル積コードへの任意精度の導入

$ git clone https://naosou@bitbucket.org/naosou/test_dynamic_precision.git
$ cd test_dynamic_precision
C言語
$ make test_c_*type* ： type → double, single, fp21 or fp42
./test_c_*type* N　：　実行時引数 N→行列の行数
Fortran
$ make test_f_*type* : type → double, single, fp21 or fp42
./test_f_*type* N　：　実行時引数 N→行列の行数

- FP21、FP42配列は作れるようにしてあるので、行列ベクトル積の部分のみ編集ください。
- 1行あたりの要素数は9です。
- 低精度化で計算時間が短くなるか、確認してください。(N=100,000,000)
- 問題サイズNで低精度化の効果がどうかわるか確認してください。
- コンパイラの変更はMakefileの先頭部分を変更ください。

Intelコンパイラ版
SYSTEM := INTEL
SYSTEM := GNU

GNUコンパイラ版
SYSTEM := INTEL
SYSTEM := GNU

任意精度ライブラリ 演習 3/3

- CRS形式について
  - ✓ 疎行列(要素のほとんどが0の行列)を表現する方法の1つ
  - ✓ row_ptr, col_ind, valの3つの配列で格納
    - row_ptr：col_ind, valの列毎の範囲を格納
    - col_ind：行番号を格納
    - val：非ゼロ要素を格納

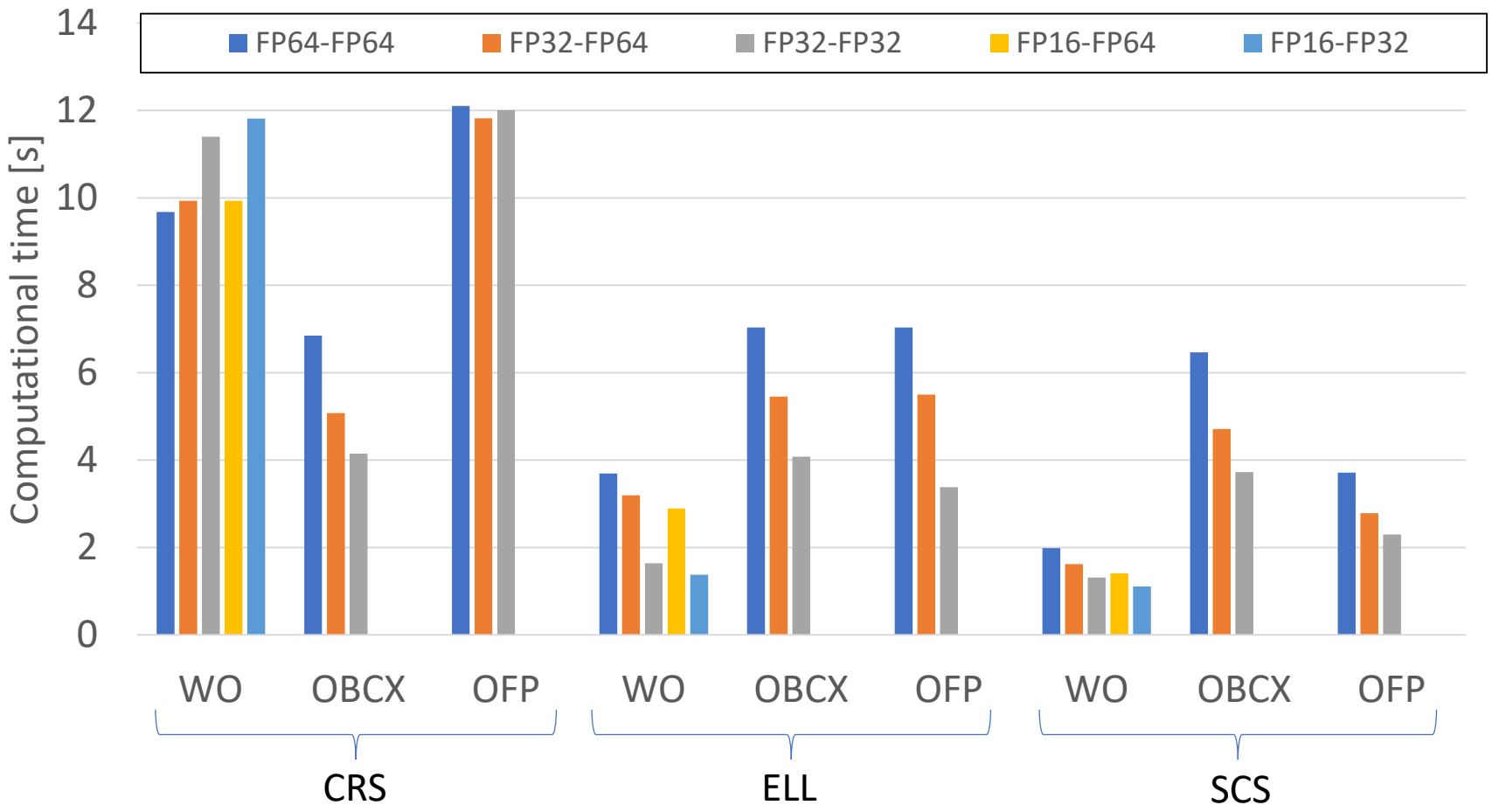
$$A = \begin{bmatrix} a & 0 & b & 0 \\ 0 & c & 0 & 0 \\ 0 & 0 & d & e \\ f & 0 & 0 & g \end{bmatrix}$$

row_ptr = (1, 3, 4, 6, 8)
col_ind = (1, 3, 2, 3, 4, 1, 4)
val = (a, b, c, d, e, f, g)

答え

編集中のディレクトリとは異なるディレクトリで、
$ git clone https://naosou@bitbucket.org/naosou/test_dynamic_precision.git
$ checkout -b answer origin/answer
src/matvec_crs_c_ans.c, src/matvec_crs_f_ans.f90が回答例

低精度化 効果一例


FP64-FP64
FP32-FP64
FP32-FP32
FP16-FP64
FP16-FP32
Computational time [s]
0
2
4
6
8
10
12
14
WO
OBCX
OFP
WO
OBCX
OFP
WO
OBCX
OFP
CRS
ELL
SCS

本講習会の内容

全3時間

- 任意精度：1時間
- 任意精度ライブラリの演習：45分
- **精度保証：1時間**
- 精度保証ライブラリの演習：15分

精度保証の必要性

低精度化が進んでいるが、解の精度は大丈夫なのか？

計算精度が足りないと・・・・。

■ 常微分方程式の場合
- ✓ 発散すればいいが、それらしい解に収束すると・・・。

■ 変微分方程式の場合
- ✓ 反復法を用いて近似解を得ているが、解が正しいかは・・・。
  - 疑似収束
    計算の過程で、ある指標に基づいて近似解を評価するが、この指標が
  - そもそも解く問題が変わった。
    低精度化によって、重要な値が欠落し、得べき問題が変わってしまった。

特化型精度保証手法の重要性

常微分方程式、直接法、反復法などに特化することで、
少ない演算量での精度保証を実現

- シンプルなアプローチは区間演算をすべての計算に適用
  - ✓ 区間演算：計算誤差を考慮して、誤差の上下限を算出
  - ✓ これをすべての計算に適用
    - → 演算が非常に重たくなる
    - → 演算回数に応じて区間が広がるため、多いと不適

- 特化型にすることで…
  - ✓ 数学的に精度を算出することで、少ない演算量で精度保証を実現

準備

$\boldsymbol{A} = (a_{i,j}) \in \mathbb{R}^{n\times n}$について、

- 行列の絶対値$|\boldsymbol{A}| = \left(\left|a_{i,j}\right|\right)$
- $\boldsymbol{A} \geq \boldsymbol{0} \rightarrow \left(a_{i,j}\right) \geq 0$
- $\|\boldsymbol{A}\|_\infty$：行列$\boldsymbol{A}$の無限大ノルム

これらはベクトルに関しても同様

ベクトルについて

- $\boldsymbol{e}$: 全ての要素が1のベクトル

誤差の区間

- $\underline{\boldsymbol{r}} \leq \boldsymbol{r} \leq \overline{\boldsymbol{r}}$

三角不等式

- $\|\boldsymbol{x} + \boldsymbol{y}\| \leq \|\boldsymbol{x}\| + \|\boldsymbol{y}\|$

精度保証対象の問題

解くべき問題の係数行列がM or H行列性を持つ場合に限定

解くべき問題: $\boldsymbol{Ax} = \boldsymbol{b}$　$\boldsymbol{A} = (a_{i,j}) \in \mathbb{R}^{n\times n}$かつ疎行列、$\boldsymbol{x} \in \mathbb{R}^n$、$\boldsymbol{b} \in \mathbb{R}^n$

今回使用した精度保証手法は単調行列を前提
大規模疎行列問題では単調行列かどうかの判断が困難
→ 簡単な方法で判断可能なM、H行列性を対象に

- ■ 単調行列
  - ✓ 正則
  - ✓ $\boldsymbol{A}^{-1} \geq \boldsymbol{O}$
- ■ M行列性
  - ✓ 単調行列
  - ✓ $a_{i,i} > 0$ かつ $a_{i,j} \leq 0,\ i \neq j$
- ■ H行列性
  - ✓ $\boldsymbol{A}$の比較行列$\langle \boldsymbol{A} \rangle := |\boldsymbol{D}| - |\boldsymbol{E}|$がM行列性を持つ
    $\boldsymbol{A} = \boldsymbol{D} + \boldsymbol{E}$、$\boldsymbol{D}$:対角行列、$\boldsymbol{E}$:非対角行列

M行列性の確認方法

1. $\tilde{\boldsymbol{y}} \leftarrow$ Solve $\boldsymbol{Ay} = \boldsymbol{e}$
2. Check $\tilde{y}_i > 0$
3. $\underline{\boldsymbol{u}} = \boldsymbol{A}\tilde{\boldsymbol{y}}$ (Rounding down)
4. Check $\underline{u}_i > 0$　ここまでを満たせばAはM行列性を持つ

疎行列線形ソルバのための精度保証

反復法から取得した近似解と真の解の最大誤差$\|\tilde{x} - x^*\|_\infty$を高精度に見積もる

$Ax = b$：解くべき方程式, x^*：真の解, $\tilde{x}$：近似解

$$\|\tilde{x} - x^*\|_\infty \leq \|\tilde{z}\|_\infty + \|A^{-1}\|_\infty \|A\tilde{z} - r\|_\infty$$

$r = b - A\tilde{x}$：残差, $Az = r$：誤差方程式, $\tilde{z}$：近似解

proof

$$\begin{aligned}
\tilde{x} - x^* &= \tilde{x} - A^{-1}b \\
&= A^{-1}(A\tilde{x} - b) \\
&= A^{-1}r + \tilde{z} - \tilde{z} \\
&= \tilde{z} - A^{-1}(A\tilde{z} - r) \\
\|\tilde{x} - x^*\|_\infty &= \|\tilde{z} - A^{-1}(A\tilde{z} - r)\|_\infty \\
&\leq \|\tilde{z}\|_\infty + \|A^{-1}\|_\infty \|A\tilde{z} - r\|_\infty
\end{aligned}$$

T. Ogita, S. Oishi & Y. Ushiro, “Computation of Sharp Rigorous Componentwise Error Bounds for the Approximate Solutions of Systems of Linear Equations.”, Reliable Computing, vol.9, p.229–239, 2003.

残差rの置換

残差を高精度で計算し、より予測誤差の小さい精度保証に

$$\begin{aligned}\|\tilde{\boldsymbol{x}} - \boldsymbol{x}^*\|_\infty &\le \|\tilde{\boldsymbol{z}}\|_\infty + \|\boldsymbol{A}^{-1}\|_\infty \|\boldsymbol{A}\tilde{\boldsymbol{z}} - \boldsymbol{r}\|_\infty \\ &\le \|\tilde{\boldsymbol{z}}\|_\infty + \|\boldsymbol{A}^{-1}\|_\infty \left(\|\boldsymbol{A}\tilde{\boldsymbol{z}} - \underline{\boldsymbol{r}}\|_\infty + \|\overline{\boldsymbol{r}} - \underline{\boldsymbol{r}}\|_\infty \right)\end{aligned}$$

この区間は小さいほど良い
→ 高精度演算で

proof

$\underline{\boldsymbol{r}} \le \boldsymbol{r} \le \overline{\boldsymbol{r}}$として、

$$\begin{aligned}\|\boldsymbol{A}\tilde{\boldsymbol{z}} - \boldsymbol{r}\|_\infty &= \|\boldsymbol{A}\tilde{\boldsymbol{z}} - \boldsymbol{r} + \underline{\boldsymbol{r}} - \underline{\boldsymbol{r}}\|_\infty \\ &\le \|\boldsymbol{A}\tilde{\boldsymbol{z}} - \underline{\boldsymbol{r}}\|_\infty + \|\boldsymbol{r} \\ &\le \|\boldsymbol{A}\tilde{\boldsymbol{z}} - \underline{\boldsymbol{r}}\|_\infty + \|\overline{\boldsymbol{r}}\end{aligned}$$

T. Ogita, S. Oishi & Y. Ushiro, “Computation of Sharp Rigorous Componentwise Error Bounds for the Approximate Solutions of Systems of Linear Equations.”, Reliable Computing, vol.9, p.229–239, 2003.

A^{-1}の置換

逆行列を置換し、係数行列が疎行列な問題で利用可能な式を算出

$$\|\tilde{\boldsymbol{x}} - \boldsymbol{x}^*\|_\infty \leq \|\tilde{\boldsymbol{z}}\|_\infty + \|\boldsymbol{A}^{-1}\|_\infty \left(\|\boldsymbol{A}\tilde{\boldsymbol{z}} - \underline{\boldsymbol{r}}\|_\infty + \|\overline{\boldsymbol{r}} - \underline{\boldsymbol{r}}\|_\infty \right)$$

$$\leq \|\tilde{\boldsymbol{z}}\|_\infty + \frac{\|\tilde{\boldsymbol{y}}\|_\infty \left(\|\boldsymbol{A}\tilde{\boldsymbol{z}} - \underline{\boldsymbol{r}}\|_\infty + \|\underline{\boldsymbol{r}} - \overline{\boldsymbol{r}}\|_\infty \right)}{1 - \|\boldsymbol{e} - \boldsymbol{A}(\tilde{\boldsymbol{y}})\|_\infty}, \boldsymbol{e} = \{1,1,1, \ldots 1\}^T$$

$$\leftarrow \|\boldsymbol{A}^{-1}\|_\infty \leq \frac{\|\tilde{\boldsymbol{y}}\|_\infty}{1 - \|\boldsymbol{e} - \boldsymbol{A}(\tilde{\boldsymbol{y}})\|_\infty}$$

proof

$\boldsymbol{A}$を単調行列、 $\boldsymbol{Ay} = \boldsymbol{e}$、 $\tilde{\boldsymbol{y}}$を近似解として、

$$|\boldsymbol{A}^{-1}\boldsymbol{e} - \tilde{\boldsymbol{y}}| = |\boldsymbol{A}^{-1}(\boldsymbol{e} - A\tilde{\boldsymbol{y}})| \leq |\boldsymbol{A}^{-1}||(\boldsymbol{e} - A\tilde{\boldsymbol{y}})|$$

$$\leq \|(\boldsymbol{e} - A\tilde{\boldsymbol{y}})\|_\infty \boldsymbol{A}^{-1}\boldsymbol{e} \because |\boldsymbol{A}^{-1}| = \boldsymbol{A}^{-1}, |\boldsymbol{g}|$$

$$(1 - \|(\boldsymbol{e} - A\tilde{\boldsymbol{y}})\|_\infty)\boldsymbol{A}^{-1}\boldsymbol{e} \leq |\tilde{\boldsymbol{y}}|$$

$$\boldsymbol{A}^{-1} \leq \frac{|\tilde{\boldsymbol{y}}|}{1 - \|\boldsymbol{e} - \boldsymbol{A}(\tilde{\boldsymbol{y}})\|_\infty}, \text{if } \|\boldsymbol{e} - \boldsymbol{A}(\tilde{\boldsymbol{y}})\|_\infty < 1$$

T. Ogita, et al. "Fast Inclusion and Residual Iteration for Solutions of Matrix Equations." Inclusion Methods for Nonlinear Problems: With Applications in Engineering, Economics and Physics, pp.171-184, 2002

精度保証手法

導出された精度保証手法は以下の通り

$$\|\tilde{\boldsymbol{x}} - \boldsymbol{x}^*\|_\infty \leq e_{abs} = \|\tilde{\boldsymbol{z}}\|_\infty + \frac{\|\tilde{\boldsymbol{y}}\|_\infty (\|\boldsymbol{A}\tilde{\boldsymbol{z}} - \dot{\boldsymbol{r}}\|_\infty + \|\dot{\boldsymbol{r}} - \bar{\boldsymbol{r}}\|_\infty)}{1 - \|\boldsymbol{e} - \boldsymbol{A}(\tilde{\boldsymbol{y}})\|_\infty}$$

$$\frac{\|\tilde{\boldsymbol{x}} - \boldsymbol{x}^*\|_\infty}{\|\boldsymbol{x}^*\|_\infty} \leq e_{rel} = \frac{e_{abs}}{\|\tilde{\boldsymbol{x}}\|_\infty - e_{abs}}$$

$\boldsymbol{x}^*$ ： 真の解

$\tilde{\boldsymbol{x}} \longrightarrow$ solved $\boldsymbol{Ax} = \boldsymbol{b}$

$\dot{\boldsymbol{r}} = \boldsymbol{b} - \boldsymbol{A}(\tilde{\boldsymbol{x}})$ (高精度演算)

$\tilde{\boldsymbol{z}} \longrightarrow$ solved $\boldsymbol{Az} = \dot{\boldsymbol{r}}$

$\tilde{\boldsymbol{y}} \longrightarrow$ solved $\boldsymbol{Ay} = \boldsymbol{e}$

$$\|\boldsymbol{e} - \boldsymbol{A}(\tilde{\boldsymbol{y}})\|_\infty = \max\left(\|\bar{\boldsymbol{s}}\|_\infty, \|\underline{\boldsymbol{s}}\|_\infty\right)$$

$\bar{\boldsymbol{s}} = -(\boldsymbol{A}(\tilde{\boldsymbol{y}}) - \boldsymbol{e})$ (Rounding up)

$\underline{\boldsymbol{s}} = -(\boldsymbol{A}(\tilde{\boldsymbol{y}}) - \boldsymbol{e})$ (Rounding down)

$\dot{\boldsymbol{r}} = \left(\bar{\boldsymbol{r}} + \underline{\boldsymbol{r}}\right) * 0.5$

$\bar{\boldsymbol{r}} = -(\boldsymbol{A}(\tilde{\boldsymbol{x}}) - \boldsymbol{b})$ (Upper bound calculated with higher precision)

$\underline{\boldsymbol{r}} = -(\boldsymbol{A}(\tilde{\boldsymbol{x}}) - \boldsymbol{b})$ (Lower bound calculated with higher precision)

Minamihata, Atsushi, et al. "Fast Verified Solutions of Sparse Linear Systems with H-matrices." Reliab. Comput. 19.2 (2013): 127-141.

精度保証のプロセス (M行列版)

1. $\tilde{\boldsymbol{x}} \leftarrow$ Solve $\boldsymbol{Ax} = \boldsymbol{b}$
2. $\tilde{\boldsymbol{y}} \leftarrow$ Solve $\boldsymbol{Ay} = \boldsymbol{e}$ $\quad \boldsymbol{e} = \{1,1,1, \ldots 1\}^T$
3. Check $\tilde{y}_i > 0$ if(false) “Verification failed”
4. $\underline{\boldsymbol{u}} = \boldsymbol{A}\tilde{\boldsymbol{y}}$ (Rounding down)
5. Check $\underline{u}_i > 0$ if(true) “$\boldsymbol{A} \in$ M-matrix” else “Verification failed”
6. $\overline{\boldsymbol{r}} = -(\boldsymbol{Ax} - \boldsymbol{b})$ (Upper bound calculated with higher precision)
7. $\underline{\boldsymbol{r}} = -(\boldsymbol{Ax} - \boldsymbol{b})$ (Lower bound calculated with higher precision)
8. $\dot{\boldsymbol{r}} = \left(\overline{\boldsymbol{r}} + \underline{\boldsymbol{r}}\right) \times 0.5$, $\boldsymbol{d} = (\dot{\boldsymbol{r}} - \overline{\boldsymbol{r}})$
9. $\tilde{\boldsymbol{z}} \leftarrow$ Solve $\boldsymbol{Az} = \dot{\boldsymbol{r}}$
10. $\overline{\boldsymbol{t}} = -(\boldsymbol{A}\tilde{\boldsymbol{z}} - \dot{\boldsymbol{r}})$, $\overline{\boldsymbol{s}} = -(\boldsymbol{A}\tilde{\boldsymbol{y}} - \boldsymbol{e})$ (Rounding up)
11. $\underline{\boldsymbol{t}} = -(\boldsymbol{A}\tilde{\boldsymbol{z}} - \dot{\boldsymbol{r}})$, $\underline{\boldsymbol{s}} = -(\boldsymbol{A}\tilde{\boldsymbol{y}} - \boldsymbol{e})$ (Rounding down)
12. $\|\boldsymbol{s}\|_\infty = \max(\|\overline{\boldsymbol{s}}\|_\infty, \|\underline{\boldsymbol{s}}\|_\infty)$, $\|\boldsymbol{t}\|_\infty = \max(\|\overline{\boldsymbol{t}}\|_\infty, \|\underline{\boldsymbol{t}}\|_\infty)$
13. $e_{abs} = \|\tilde{\boldsymbol{z}}\|_\infty + \frac{\|\tilde{\boldsymbol{y}}\|_\infty (\|\boldsymbol{t}\|_\infty + \|\boldsymbol{d}\|_\infty)}{1.0 - \|\boldsymbol{s}\|_\infty}$,
14. $e_{rel} = \left\| \frac{e_{abs}}{|\tilde{x}| - e_{abs}} \right\|_\infty$

1はそもそも解いている部分
2は緩い収束条件でOK
9は1と同程度の収束条件が必要

精度保証のプロセス (H行列版)

1. $\tilde{\boldsymbol{x}} \leftarrow$ Solve $\boldsymbol{Ax} = \boldsymbol{b}$
2. $\tilde{\boldsymbol{y}} \leftarrow$ Solve $\langle \boldsymbol{A} \rangle \boldsymbol{y} = \boldsymbol{e} \quad \boldsymbol{e} = \{1,1,1, \ldots 1\}^T$
3. Check $\tilde{y}_i > 0$ if(false) “Verification failed”
4. $\underline{\boldsymbol{u}} = \langle \boldsymbol{A} \rangle \tilde{\boldsymbol{y}}$ (Rounding down)
5. Check $\underline{u}_i > 0$ if(true) “$\boldsymbol{A} \in$ H-matrix” else “Verification failed”
6. $\overline{\boldsymbol{r}} = -(\boldsymbol{Ax} - \boldsymbol{b})$ (Upper bound calculated with higher precision)
7. $\underline{\boldsymbol{r}} = -(\boldsymbol{Ax} - \boldsymbol{b})$ (Lower bound calculated with higher precision)
8. $\dot{\boldsymbol{r}} = \left(\overline{\boldsymbol{r}} + \underline{\boldsymbol{r}}\right) \times 0.5, \boldsymbol{d} = (\dot{\boldsymbol{r}} - \overline{\boldsymbol{r}})$
9. $\tilde{\boldsymbol{z}} \leftarrow$ Solve $\boldsymbol{Az} = \dot{\boldsymbol{r}}$
10. $\overline{\boldsymbol{t}} = -(\boldsymbol{A}\tilde{\boldsymbol{z}} - \dot{\boldsymbol{r}}), \overline{\boldsymbol{s}} = -(\langle \boldsymbol{A} \rangle \tilde{\boldsymbol{y}} - \boldsymbol{e})$ (Rounding up)
11. $\underline{\boldsymbol{t}} = -(\boldsymbol{A}\tilde{\boldsymbol{z}} - \dot{\boldsymbol{r}}), \underline{\boldsymbol{s}} = -(\langle \boldsymbol{A} \rangle \tilde{\boldsymbol{y}} - \boldsymbol{e})$ (Rounding down)
12. $\|\boldsymbol{s}\|_\infty = \max(\|\overline{\boldsymbol{s}}\|_\infty, \|\underline{\boldsymbol{s}}\|_\infty), \|\boldsymbol{t}\|_\infty = \max(\|\overline{\boldsymbol{t}}\|_\infty, \|\underline{\boldsymbol{t}}\|_\infty)$
13. $e_{abs} = \|\tilde{\boldsymbol{z}}\|_\infty + \frac{\|\tilde{\boldsymbol{y}}\|_\infty (\|\boldsymbol{t}\|_\infty + \|\boldsymbol{d}\|_\infty)}{1.0 - \|\boldsymbol{s}\|_\infty}$
14. $e_{rel} = \left\| \frac{e_{abs}}{|\tilde{\boldsymbol{x}}| - e_{abs}} \right\|_\infty$

精度保証のプロセスの計算時間

精度保証のオーバーヘッドはおおよそ2～3倍？

1. $\tilde{\boldsymbol{x}} \leftarrow$ Solve $\boldsymbol{Ax} = \boldsymbol{b}$
2. $\tilde{\boldsymbol{y}} \leftarrow$ Solve $\boldsymbol{Ay} = \boldsymbol{e}$ $\quad \boldsymbol{e} = \{1,1,1, \ldots 1\}^T$ (1.より大分軽い)
3. Check $\tilde{y}_i > 0$ if(false) “Verification failed”
4. $\underline{\boldsymbol{u}} = \boldsymbol{A}\tilde{\boldsymbol{y}}$ (Rounding down)
5. Check $\underline{u_i} > 0$ if(true) “$\boldsymbol{A} \in$ M-matrix” else “Verification failed”
6. $\overline{\boldsymbol{r}} = -(\boldsymbol{Ax} - \boldsymbol{b})$ (Upper bound calculated with higher precision)
7. $\underline{\boldsymbol{r}} = -(\boldsymbol{Ax} - \boldsymbol{b})$ (Lower bound calculated with higher precision)
8. $\dot{\boldsymbol{r}} = \left(\overline{\boldsymbol{r}} + \underline{\boldsymbol{r}}\right) \times 0.5, \boldsymbol{d} = (\dot{\boldsymbol{r}} - \overline{\boldsymbol{r}})$
9. $\tilde{\boldsymbol{z}} \leftarrow$ Solve $\boldsymbol{Az} = \dot{\boldsymbol{r}}$ (1.と同程度)
10. $\overline{\boldsymbol{t}} = -(\boldsymbol{A}\tilde{\boldsymbol{z}} - \dot{\boldsymbol{r}}), \overline{\boldsymbol{s}} = -(\boldsymbol{A}\tilde{\boldsymbol{y}} - \boldsymbol{e})$ (Rounding up)
11. $\underline{\boldsymbol{t}} = -(\boldsymbol{A}\tilde{\boldsymbol{z}} - \dot{\boldsymbol{r}}), \underline{\boldsymbol{s}} = -(\boldsymbol{A}\tilde{\boldsymbol{y}} - \boldsymbol{e})$ (Rounding down)
12. $\|\boldsymbol{s}\|_\infty = \max(\|\overline{\boldsymbol{s}}\|_\infty, \|\underline{\boldsymbol{s}}\|_\infty), \|\boldsymbol{t}\|_\infty = \max(\|\overline{\boldsymbol{t}}\|_\infty, \|\underline{\boldsymbol{t}}\|_\infty)$
13. $e_{abs} = \|\tilde{\boldsymbol{z}}\|_\infty + \frac{\|\tilde{\boldsymbol{y}}\|_\infty (\|\boldsymbol{t}\|_\infty + \|\boldsymbol{d}\|_\infty)}{1.0 - \|\boldsymbol{s}\|_\infty}$
14. $e_{rel} = \left\| \frac{e_{abs}}{|\tilde{\boldsymbol{x}}| - e_{abs}} \right\|_\infty$

ライブラリの利用方法

3パターンの利用方法を提供

ライブラリでは3つの利用方法をサポート(1以外はC、Fortran両対応)準備中

1. 行列、ベクトルのファイルを入力
   - とりあえず解きたい問題の精度を確認するための方法
   - 行列の性質のみで誤差を算出
     - ✓ ソルバ部は講師実装
   - IC(t)CGソルバ(公開積み別コード)に精度保証を導入
2. ユーザーのソルバを呼び出す
   - サンプルコード的な立ち位置
   - ソルバの実装も考慮した誤差を算出
     - ✓ ソルバ以外はライブラリが担当
   - 実装中
3. 任意のソルバに埋め込む
   - HPC向け
   - ライブラリは最低限の機能のみ提供
     - ✓ 行列ベクトル積もユーザー側
     - ✓ Rounding down, upの切り替えや高精度演算をライブラリが担当
   - OpenMP、MPI対応

演習では1、2を実施いただきます。

利用方法1

とりあえず行列の性質を確認したい場合に有効

- IC(t)CGソルバに精度保証ライブラリ(利用方法3)を取り込んだコードを公開中
- ファイルで行列、ベクトルを取り込み
  - ✓ Matrix Market形式に対応
- Fortran＋OpenMP

閾値付きICCG法

IC(t)CG法と表記

- 英語の呼び名はdual threshold ICCG (最大fill-inレベルと閾値を使うため)
  - ✓ コレスキー分解の過程で0要素が0でなくなるのがfill-in
  - ✓ 発生したfill-inのレベルと閾値が条件を満たせば残す、それ以外は破棄

Algorithm of IC(t)CG

do k = 1, until converge

$$\alpha = \frac{(\boldsymbol{r}^k, \boldsymbol{p}^k)}{(\boldsymbol{p}, ^k \boldsymbol{A}\boldsymbol{p}^k)}$$

$$\boldsymbol{x}^{k+1} = \boldsymbol{x}^k + \alpha \boldsymbol{p}^k$$

$\boldsymbol{q}^k$: vector shows searching direction

$$\boldsymbol{r}^{k+1} = \boldsymbol{r}^k - \alpha \boldsymbol{A}\boldsymbol{p}^k$$

$\boldsymbol{r}^k$: residual vector

$$\acute{\boldsymbol{r}} = \boldsymbol{r}^k$$

$$\boldsymbol{q} = \acute{\boldsymbol{U}}^{-1}\acute{\boldsymbol{D}}^{-1}\acute{\boldsymbol{L}}^{-1}\acute{\boldsymbol{r}}$$

$$\boldsymbol{p}^{k+1} = \boldsymbol{q} - \frac{(\boldsymbol{q}, \boldsymbol{r}^{k+1})}{\rho}\boldsymbol{p}^k, \rho = (\boldsymbol{q}, \boldsymbol{r}^{k+1})$$

enddo

利用方法2 1/2

問題を解いている部分をユーザー関数に変更

赤:ユーザー実装関数

1. $\tilde{\boldsymbol{x}} \leftarrow$ Solve $\boldsymbol{Ax} = \boldsymbol{b}$
2. $\tilde{\boldsymbol{y}} \leftarrow$ Solve $\boldsymbol{Ay} = \boldsymbol{e}$ $\quad \boldsymbol{e} = \{1,1,1, \ldots 1\}^T$
3. Check $\tilde{y}_i > 0$ if(false) “Verification failed”
4. $\underline{\boldsymbol{u}} = \boldsymbol{A}\tilde{\boldsymbol{y}}$ (Rounding down)
5. Check $\underline{u}_i > 0$ if(true) “$\boldsymbol{A} \in$ M-matrix” else “Verification failed”
6. $\overline{\boldsymbol{r}} = -(\boldsymbol{Ax} - \boldsymbol{b})$ (Upper bound calculated with higher precision)
7. $\underline{\boldsymbol{r}} = -(\boldsymbol{Ax} - \boldsymbol{b})$ (Lower bound calculated with higher precision)
8. $\dot{\boldsymbol{r}} = \left(\overline{\boldsymbol{r}} + \underline{\boldsymbol{r}}\right) \times 0.5$, $\boldsymbol{d} = (\dot{\boldsymbol{r}} - \overline{\boldsymbol{r}})$
9. $\tilde{\boldsymbol{z}} \leftarrow$ Solve $\boldsymbol{Az} = \dot{\boldsymbol{r}}$
10. $\overline{\boldsymbol{t}} = -(\boldsymbol{A}\tilde{\boldsymbol{z}} - \dot{\boldsymbol{r}})$, $\overline{\boldsymbol{s}} = -(\boldsymbol{A}\tilde{\boldsymbol{y}} - \boldsymbol{e})$ (Rounding up)
11. $\underline{\boldsymbol{t}} = -(\boldsymbol{A}\tilde{\boldsymbol{z}} - \dot{\boldsymbol{r}})$, $\underline{\boldsymbol{s}} = -(\boldsymbol{A}\tilde{\boldsymbol{y}} - \boldsymbol{e})$ (Rounding down)
12. $\|\boldsymbol{s}\|_\infty = \max(\|\overline{\boldsymbol{s}}\|_\infty, \|\underline{\boldsymbol{s}}\|_\infty)$, $\|\boldsymbol{t}\|_\infty = \max(\|\overline{\boldsymbol{t}}\|_\infty, \|\underline{\boldsymbol{t}}\|_\infty)$
13. $e_{abs} = \|\tilde{\boldsymbol{z}}\|_\infty + \frac{\|\tilde{\boldsymbol{y}}\|_\infty(\|\boldsymbol{t}\|_\infty + \|\boldsymbol{d}\|_\infty)}{1.0 - \|\boldsymbol{s}\|_\infty}$
14. $e_{rel} = \left\| \frac{e_{abs}}{|\tilde{\boldsymbol{x}}| - e_{abs}} \right\|_\infty$

利用方法2 2/2

問題を解いている部分をユーザー関数に変更

1. $\tilde{\boldsymbol{x}} \leftarrow$ Solve $\boldsymbol{Ax} = \boldsymbol{b}$
2. init_verify(row_ptr, col_ind, val, n_row, st)
3. $\tilde{\boldsymbol{y}} \leftarrow$ Solve $\boldsymbol{Ay} = \boldsymbol{e}$ $\quad \boldsymbol{e} = \{1,1,1, \ldots 1\}^T$
4. error = verify_intermid($\tilde{\boldsymbol{x}}$, $\tilde{\boldsymbol{y}}$, $\dot{\boldsymbol{r}}$, st1)
5. $\tilde{\boldsymbol{z}} \leftarrow$ Solve $\boldsymbol{Az} = \dot{\boldsymbol{r}}$
6. error = verify_finalize($\tilde{\boldsymbol{z}}$, ae, re, st1)

赤：ユーザー実装関数
青：ライブラリ関数

row_ptr, col_ind, val : 行列AのCRS形式
配列x, y, r, zの確保はユーザー
ae : Absoute error
re : Relative error

利用方法3 1/2

行列ベクトル積を等、性能に影響する部分はユーザー関数

1. $\tilde{\boldsymbol{x}} \leftarrow$ Solve $\boldsymbol{Ax} = \boldsymbol{b}$
2. $\tilde{\boldsymbol{y}} \leftarrow$ Solve $\boldsymbol{Ay} = \boldsymbol{e}$ $\quad \boldsymbol{e} = \{1,1,1, \ldots 1\}^T$
3. Check $\tilde{y}_i > 0$ if(false) “Verification failed”
4. $\underline{\boldsymbol{u}} = \boldsymbol{A}\tilde{\boldsymbol{y}}$ (Rounding down)
5. Check $\underline{u_i} > 0$ if(true) “$\boldsymbol{A} \in$ M-matrix” else “Verification failed”
6. $\overline{\boldsymbol{r}} = -(\boldsymbol{Ax} - \boldsymbol{b})$ (Upper bound calculated with higher precision)
7. $\underline{\boldsymbol{r}} = -(\boldsymbol{Ax} - \boldsymbol{b})$ (Lower bound calculated with higher precision)
8. $\dot{\boldsymbol{r}} = \left(\overline{\boldsymbol{r}} + \underline{\boldsymbol{r}}\right) \times 0.5$, $\boldsymbol{d} = (\dot{\boldsymbol{r}} - \overline{\boldsymbol{r}})$
9. $\tilde{\boldsymbol{z}} \leftarrow$ Solve $\boldsymbol{Az} = \dot{\boldsymbol{r}}$
10. $\overline{\boldsymbol{t}} = -(\boldsymbol{A}\tilde{\boldsymbol{z}} - \dot{\boldsymbol{r}})$, $\overline{\boldsymbol{s}} = -(\boldsymbol{A}\tilde{\boldsymbol{y}} - \boldsymbol{e})$ (Rounding up)
11. $\underline{\boldsymbol{t}} = -(\boldsymbol{A}\tilde{\boldsymbol{z}} - \dot{\boldsymbol{r}})$, $\underline{\boldsymbol{s}} = -(\boldsymbol{A}\tilde{\boldsymbol{y}} - \boldsymbol{e})$ (Rounding down)
12. $\|\boldsymbol{s}\|_\infty = \max(\|\overline{\boldsymbol{s}}\|_\infty, \|\underline{\boldsymbol{s}}\|_\infty)$, $\|\boldsymbol{t}\|_\infty = \max(\|\overline{\boldsymbol{t}}\|_\infty, \|\underline{\boldsymbol{t}}\|_\infty)$
13. $e_{abs} = \|\tilde{\boldsymbol{z}}\|_\infty + \frac{\|\tilde{\boldsymbol{y}}\|_\infty (\|\boldsymbol{t}\|_\infty + \|\boldsymbol{d}\|_\infty)}{1.0 - \|\boldsymbol{s}\|_\infty}$
14. $e_{rel} = \left\| \frac{e_{abs}}{|\tilde{x}| - e_{abs}} \right\|_\infty$

利用方法3 2/2

行列ベクトル積を等、性能に影響する部分はユーザー関数

1. $\tilde{x} \leftarrow$ Solve $Ax = b$
2. $\tilde{y} \leftarrow$ Solve $Ay = e$ $\quad e = \{1,1,1, \ldots 1\}^T$
3. pre_check_MH(y)
4. $\underline{u} = A\tilde{y}$
5. post_check_MH(u)
6. $\dot{r}(i) = -(Ax - b)$ (積和算 → muladd_verify(val, x, y(i), rem, err), Ax-b → post_kernel_verify(y(i), rem ,err, $\dot{r}(i)$))
7. $\tilde{z} \leftarrow$ Solve $Az = \dot{r}$
8. pre_verification(1)
9. $\overline{t} = -(A\tilde{z} - \dot{r})$, $\overline{s} = -(A\tilde{y} - e)$
10. pre_verification(2)
11. $\underline{t} = -(A\tilde{z} - \dot{r})$, $\underline{s} = -(A\tilde{y} - e)$
12. ar = post_verification($\tilde{x}$, $\tilde{y}$, $\tilde{z}$, $\dot{r}$, $\overline{t}$, $\underline{t}$, $\overline{s}$, $\underline{s}$, re)

https://naosou@bitbucket.org/naosou/ictcg.git をご参考ください。

本講習会の内容

全3時間

- 任意精度：1時間
- 任意精度ライブラリの演習：45分
- 精度保証：1時間
- **精度保証ライブラリの演習：15分**

演習

利用方法1を実施

利用方法1の演習 1/2

```
$ git clone https://naosou@bitbucket.org/naosou/lib_verify.git
$ git clone https://naosou@bitbucket.org/naosou/ictcg.git
$ git checkout -b verify_publish origin/verify_publish
$ cd src
$ ln -s ../lib_verify/kernel_verify.f90 ./
$ ln -s ../lib_verify/precision_verify.inc ./
$ ln -s ../lib_verify/verification_lowlevel.f90 ./
$ cd ../
$ make       #バイナリはrunディレクトリの配下に生成
$ cd run
$ wget https://suitesparse-collection-
website.herokuapp.com/MM/HB/685_bus.tar.gz
$ tar xfvz 685_bus.tar.gz
$ ./ICTCG_dd 685_bus/685_bus.mtx -r const -m ICTCG_SCS -f tol
1.0d-20 -f max 2 -S Chunk 256 -S sigma 1024 -c greedy
$ ./ICTCG_ss 685_bus/ 685_bus.mtx -r const -m ICTCG_SCS -f tol
1.0d-20 -f max 2 -S Chunk 256 -S sigma 1024 -c greedy
```

利用方法1の演習 2/2

./ICTCG_dd 685_bus/685_bus.mtx -r const -m ICTCG_SCS -f tol 1.0d-20 -f max 2 -S Chunk 256 -S sigma 1024 -c greedy

色が薄い部分はソルバのパラメータ
とりあえずそのままでご利用ください。

-r const
$\boldsymbol{b} = \boldsymbol{Ae}$で右辺ベクトルを算出
-r file hogehoge.mm
Matrix Market形式のベクトルも取り込み可能

利用方法2の演習

評価条件

計算機環境（Flow Type1（A64fx））

- Fujitsu Fortran Compiler 4.11.1 tcsds-1.2.39
- -O3 -Kfast,lto,openmp,zfill,A64FX,ARMV8_A,ocl,noalias=s,noeval -X03

問題設定

- $\boldsymbol{b} = \boldsymbol{Ae}$として右辺ベクトルを決定 (いずれの精度のソルバでもFP64を使って算出)
  - 解が自明なので、真の誤差を算出可能
- 各箇所での収束条件は以下(いずれも相対残差が指定値以下)、かつ解ベクトルの初期値=**0**
  - $\tilde{\boldsymbol{x}} \leftarrow$ Solve $\boldsymbol{Ax} = \boldsymbol{b}$ ： 10^{-12}
  - $\tilde{\boldsymbol{y}} \leftarrow$ Solve $\boldsymbol{Ay} = \boldsymbol{e}$ ： 10^{-2}
  - $\tilde{\boldsymbol{z}} \leftarrow$ Solve $\boldsymbol{Az} = \dot{\boldsymbol{r}}$ ： 10^{-9}
- その他反復法の条件
  - IC(2)を採用
  - 48スレッド
  - Coloring Algorithm = Greedy
- 精度の組み合わせ
  - ✓ DD (行列：FP64、ベクトル：FP64)
  - ✓ SD (行列：FP32、ベクトル：FP64)
  - ✓ SS (行列：FP32、ベクトル：FP32)
  - ✓ SSD (行列：FP32、ベクトル：FP32、全ての計算は倍精度に)

評価対象の行列

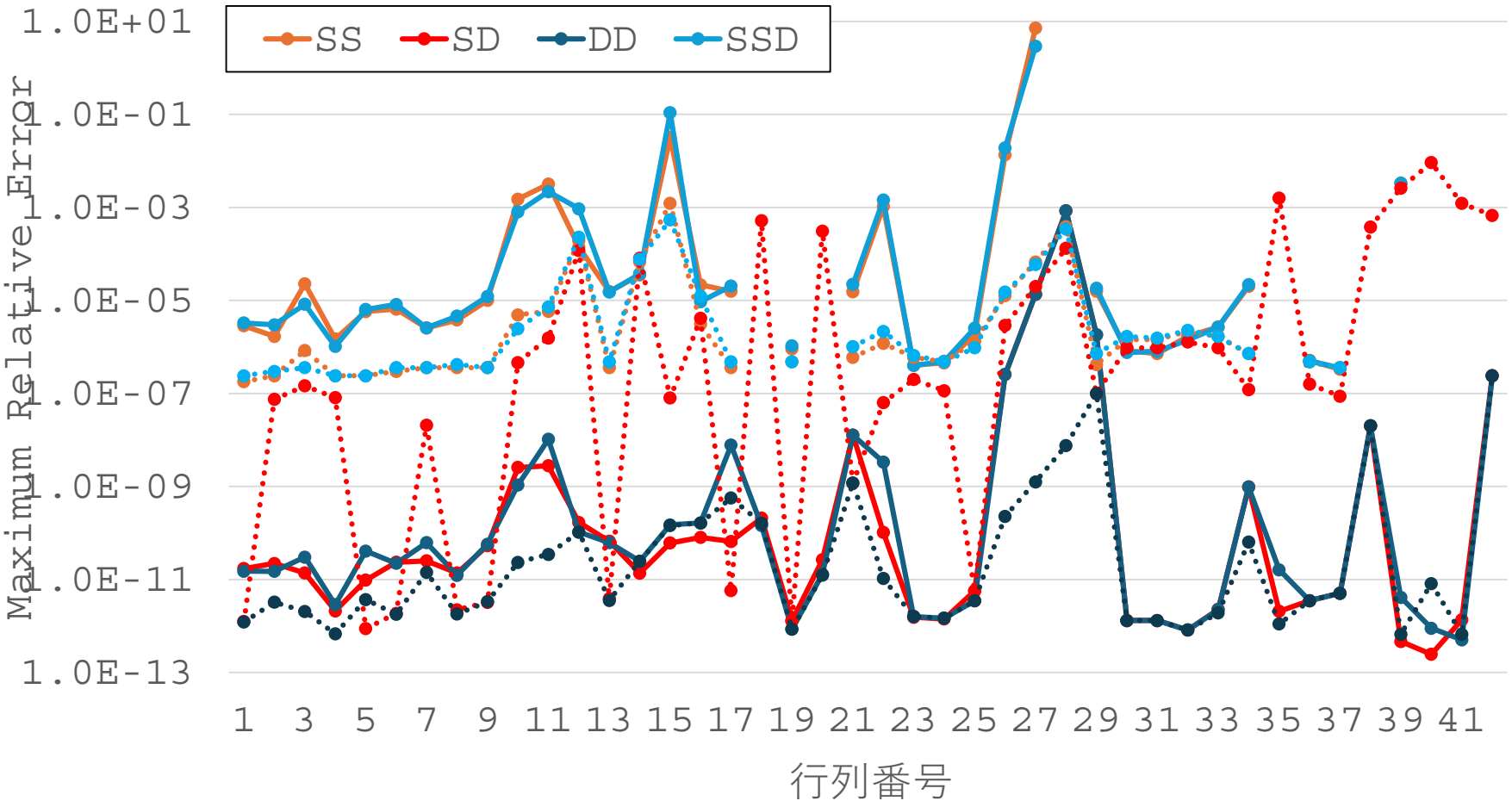
行列名	性質	DoF
Trefethen_20.mtx	H	20
mesh1e1.mtx	H	48
mesh1em1.mtx	H	48
mesh1em6.mtx	H	48
Trefethen_150.mtx	H	150
Trefethen_200.mtx	H	200
mesh3em5.mtx	H	289
mesh3e1.mtx	H	289
Trefethen_300.mtx	H	300
mesh2em5.mtx	H	306
mesh2e1.mtx	H	306
494_bus.mtx	M	494
Trefethen_500.mtx	H	500
662_bus.mtx	M	662
nos6.mtx	M	675
685_bus.mtx	M	685
Trefethen_700.mtx	H	700
nos7.mtx	M	729
gr_30_30.mtx	M	900
1138_bus.mtx	M	1138
Trefethen_2000.mtx	H	2000

行列名	性質	DoF
Chem97ZtZ.mtx	H	2541
fv1.mtx	M	9604
fv2.mtx	M	9801
fv3.mtx	M	9801
bodyy4.mtx	H	17546
bodyy5.mtx	H	18589
bodyy6.mtx	H	19366
Trefethen_20000.mtx	H	20000
obstclae.mtx	M	40000
torsion1.mtx	M	40000
jnlbrng1.mtx	M	40000
minsurfo.mtx	H	40806
finan512.mtx	H	74752
apache1.mtx	M	80800
shallow_water2.mtx	M	81920
shallow_water1.mtx	M	81920
G2_circuit.mtx	M	150102
parabolic_fem.mtx	M	525825
apache2.mtx	M	715176
ecology2.mtx	H	999999
G3_circuit.mtx	M	1585478

ここでの性質は倍精度演算で評価した結果
→精度によっては性質が変わるパターンあり

行列の性質(M or H)は倍精度演算時の結果
青：低精度化時に喪失
赤：低精度化時に変化

評価結果


SS
SD
DD
SSD
1.0E+01
1.0E-01
1.0E-03
1.0E-05
1.0E-07
1.0E-09
1.0E-11
1.0E-13
Maximum Relative Error
1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41
行列番号

お疲れ様でした！

アンケートにご協力いただければ幸甚です。

第237回講習会「変動精度数値ライブラリ」