エラーフリー変換と口バストな行列分解

荻田 武史
東京女子大学 現代教養学部 数理科学科

日本数学会年会
2010年3月

講演の概要

数学の定理等の証明をするために，コンピュータによる計算を用いる方法を「計算機援用証明（computer-assisted proof）」と呼ぶ．計算機援用証明を支える技術のひとつとして，「精度保証付き数値計算」がある．

この講演の前半では，精度保証付き数値計算における新しい基盤技術である「エラーフリー変換」について説明する．

後半は，エラーフリー変換の応用として，ロバストな行列分解アルゴリズムを紹介する．

前半の講演内容

1. 概要
2. 浮動小数点演算のerror-free変換
3. 任意計算精度の総和・内積計算
4. 任意結果精度の総和・内積計算

joint work with
Prof. Siegfried M. Rump and Prof. Shin'ichi Oishi

概要

一般的に，浮動小数点演算によって得られた結果は，丸め誤差の影響で，しばしば信頼性がなくなることはよく知られている．

目的 ベクトルの総和 $\sum_{i=1}^{n} x_i$ や内積 $x^T y = \sum_{i=1}^{n} x_i\, y_i$ を浮動小数点演算のみで高精度に求める．

$\Longrightarrow$ **科学技術計算の基本のひとつ**

応用 連立一次方程式，固有値問題，特異値問題等，それらの数値解の反復改良

通常の総和計算アルゴリズム（$\sum_{i=1}^{n} p_i$）

$$
\begin{aligned}
&s_0 = 0;\\
&for \quad i = 1 : n\\
&\qquad s_i = s_{i-1} \oplus p_i; \qquad \% \oplus\text{: fl-pt add}\\
&end
\end{aligned}
$$

このとき，丸め誤差解析から，s_n の相対誤差は

$$\left| \frac{\sum p_i - s_n}{\sum p_i} \right| \leq \mathcal{O}(\mathbf{u}) \mathrm{cond}(\sum p_i)$$

ただし，$\mathbf{u}$ は計算精度，$\mathrm{cond}(\sum p_i)$ は問題の難しさを表す条件数．

背景

ベクトルの総和・内積の高精度なアルゴリズムはたくさんある(次頁)が，大別すると以下のようなタイプに分類できる．

a) 経験的に結果の精度が改善される方式
b) 内部演算が多倍長演算のように高精度になる方式 (precision)
c) 指定した精度の結果が得られる方式 (accuracy)

多くのアルゴリズムは b) に属する．

ベクトルの総和・内積計算についての歴史（抜粋）

1965 Møller (BIT)
1970 Nickel (ZAMM) (Kahan-Babuška's algorithm)
1971 Malcolm (Comm. ACM)
1972 Pichat (Numer. Math.)
1973 Kiełbaszíński (Math. Stos.)
1974 Neumaier (ZAMM) (improved Kahan-Babuška's algorithm)
1977 Bohlender (IEEE Trans. Comput.)
1986 Kulisch, Miranker (SIAM Review, originally 1970's in Karlsruhe)
1991 Priest (Proc. IEEE Symposium)
1999 Anderson (SIAM J. Sci. Comput.)
2002 Li et al. (ACM TOMS, XBLAS, based on Bailey's DD)
2003 Demmel, Hida (SIAM J. Sci. Comput)
2005 Ogita, Rump, Oishi (SIAM J. Sci. Comput.)
2008 Rump, Ogita, Oishi (SIAM J. Sci. Comput.)

precision（演算精度）とaccuracy（結果精度）

b)に属するアルゴリズムで得られた結果を $\mathrm{res} \in \mathbb{F}$ とすると

$$\frac{|\mathrm{res} - \sum p_i|}{|\sum p_i|} \leq \mathbf{u}_{\mathrm{out}} + \mathbf{u}_{\mathrm{int}} \mathrm{cond}(\sum p_i)$$

但し，$\mathbf{u}_{\mathrm{int}}$ は内部計算精度，$\mathbf{u}_{\mathrm{out}}$ は出力フォーマットの精度，

$$\mathrm{cond}(\sum p_i) := \frac{\sum |p_i|}{|\sum p_i|}$$ (ベクトルの総和の条件数)

$\Longrightarrow$ $\mathbf{u}_{\mathrm{int}}$ precision (多倍長精度に対応．結果精度の保証はない)

precisionとaccuracy (2)

c)に属するアルゴリズムで得られた結果を $\mathrm{res} \in \mathbb{F}$ とすると

$$\frac{|\mathrm{res} - \sum p_i|}{|\sum p_i|} \leq \mathbf{u}_{\mathrm{tol}}$$

但し，$\mathbf{u}_{\mathrm{tol}}$ は指定した許容誤差

$\Longrightarrow$ $\mathbf{u}_{\mathrm{tol}}$ accuracy (条件数に依存しない．結果精度が保証されている)

計算速度の点で従来方式の不利な点

- 入力データのソートが必要
  - i) 絶対値の大きい(小さい)順 ($\mathcal{O}(n \log n)$ operations)
  - ii) 指数部の大きい（小さい）順 ($\mathcal{O}(n)$ operations)
- 内部ループが分岐を含む (less compiler optimization)
- 通常使用する精度(単精度・倍精度)の他に，より高精度なフォーマット(拡張倍精度など)が必要 (less portability)
- 仮数部や指数部へのアクセスが必要 (less portability)

本研究の方針

- 通常 (単精度・倍精度) の浮動小数点演算による和・差・積のみを用いる

- 内部ループに分岐や仮数部・指数部へのアクセス等がない

$\Longrightarrow$ 特別な高精度フォーマットや特別なハードウェアは必要ない
$\Longrightarrow$ これらの特徴から，提案手法は計算量の意味だけでなく実行時間の意味でも大変高速になる

エラーフリー変換

$\mathbb{F}$：倍精度浮動小数点数の集合
$\mathbf{u}$：浮動小数点演算の相対精度（IEEE 754倍精度では$\mathbf{u} = 2^{-53}$）

たとえば，$a, b \in \mathbb{F}$に対して，次のような$x, y \in \mathbb{F}$を得られる．

Knuth (1969) $[x, y] = \mathbf{TwoSum}(a, b) \quad \Rightarrow \quad x + y = a + b$

Dekker (1971) $[x, y] = \mathbf{TwoProduct}(a, b) \quad \Rightarrow \quad x + y = a \times b$

Error-free! (情報の欠落がない)

TwoSum (Knuth, 1969)

$$[x, y] = \textbf{TwoSum}(a, b) \quad \Rightarrow \quad a + b = x + y \quad (|y| \leq \mathbf{u}\,|x|)$$

$\oplus, \ominus$: floating point addition and subtraction

function $[x, y] = \textbf{TwoSum}(a, b)$
$x = a \oplus b;$
$c = x \ominus a;$
$y = (a \ominus (x \ominus c)) \oplus (b \ominus c);$

Computational cost: 6 flops

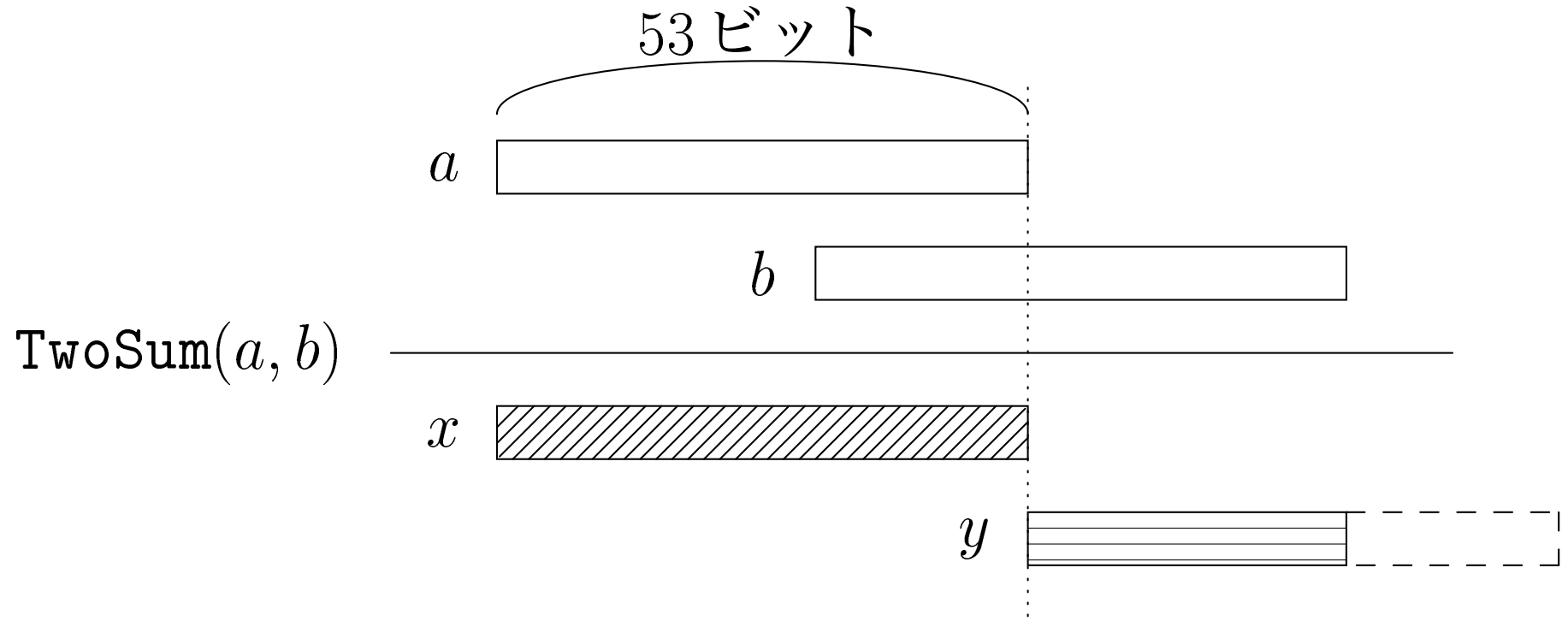


Figure 1: アルゴリズムTwoSumのイメージ．長方形が浮動小数点数を表し，左側にあるほど絶対値が大きいとする（この図では，$|a| > |b|$）．

TwoSum' (Dekker, 1971)

$$[x, y] = \textbf{TwoSum}'(a, b) \quad \Rightarrow \quad a + b = x + y \quad (|y| \leq \mathbf{u}\,|x|)$$

function $[x, y] = \textbf{TwoSum}'(a, b)$
$x = a \oplus b;$
if $|a| \geq |b|$
  $y = b \ominus (x \ominus a);$
else
  $y = a \ominus (x \ominus b);$

3 flops + taking 2 absolute values + 1 comparison

$\Longrightarrow$ **Computational speed slows down**

Split (Dekker, 1971)

$$[a_H, a_L] = \textbf{Split}(a) \quad \Rightarrow \quad a = a_H + a_L \quad (|a_H| \geq |a_L|)$$

$\otimes$: floating point multiplication

function $[a_H, a_L] = \textbf{Split}(a)$

$c = \texttt{factor} \otimes a$; % $\texttt{factor} = 2^{\lceil \frac{t}{2} \rceil} + 1$

$a_H = c \ominus (c \ominus a)$;

$a_L = a \ominus a_H$;

4 flops

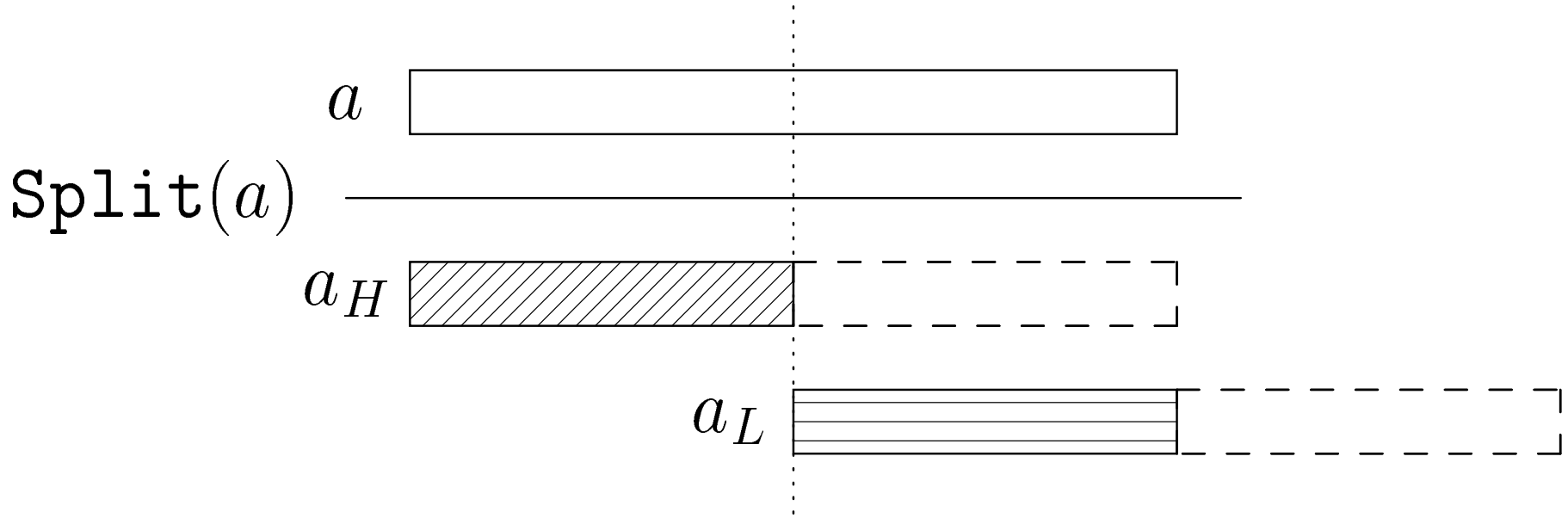


Figure 2: アルゴリズムSplitのイメージ.

TwoProduct (Veltkamp [Dekker, 1971])

$$[x, y] = \textbf{TwoProduct}(a, b) \quad \Rightarrow \quad a \times b = x + y \quad (|y| \leq \mathbf{u}\,|x|)$$

$$\begin{aligned}
&\textbf{function} \quad [x, y] = \textbf{TwoProduct}(a, b) \\
&x = a \otimes b; \\
&[a_H, a_L] = \textbf{Split}(a); \qquad \% \; a_H + a_L \leftarrow a \\
&[b_H, b_L] = \textbf{Split}(b); \qquad \% \; b_H + b_L \leftarrow b \\
&y = a_L \otimes b_L \ominus (((x \ominus a_H \otimes b_H) \ominus a_L \otimes b_H) \ominus a_H \otimes b_L);
\end{aligned}$$

17 flops

TwoProductFMA

FMA: Fused Multiply-Add (standardized by IEEE 754-2008)
By FMA(a, b, c), $d := a * b + c$ can be calculated in 1 flop.
Result d is rounded to the nearest floating point number.

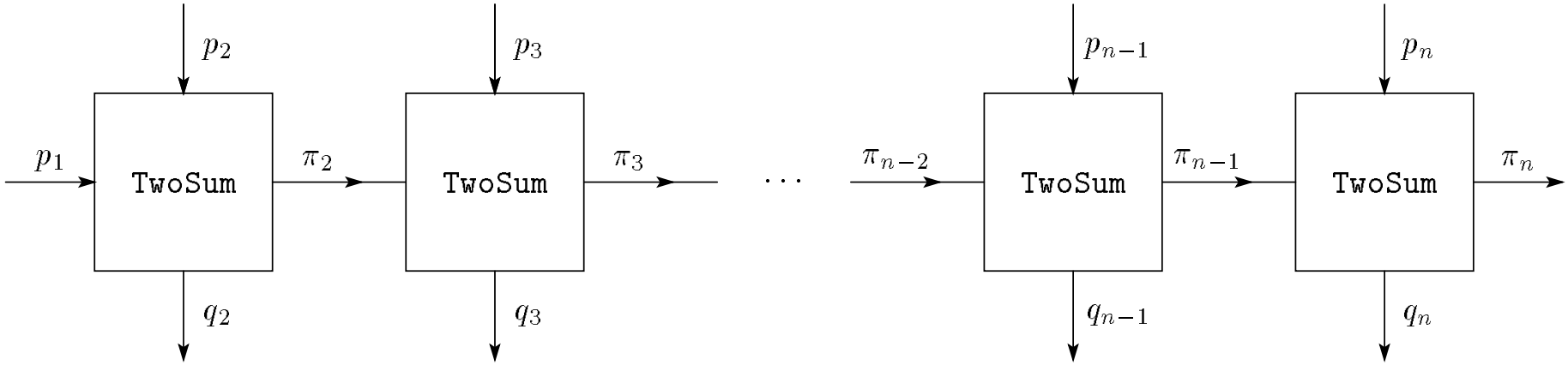
function $[x, y] =$ **TwoProductFMA**(a, b)
$x = a \otimes b;$
$y =$ **FMA**$(a, b, -x);$

17 flops $\Longrightarrow$ 2 flops

ベクトルの総和のerror-free変換

For $p \in \mathbb{F}^n$, we can obtain the following $p' \in \mathbb{F}^n$:

$$p' = \mathbf{VecSum}(p) \quad \Longrightarrow \quad \pi_n + \sum_{i=2}^{n} q_i = \sum_{i=1}^{n} p_i$$



VecSum

$$p' = \textbf{VecSum}(p) \quad \Rightarrow \quad \sum_{i=1}^{n} p_i = \sum_{i=1}^{n} p'_i$$

(so-called distillation algorithm)

function $p = \textbf{VecSum}(p)$
for $i = 2 : n$
$[p_i, p_{i-1}] = \textbf{TwoSum}(p_i, p_{i-1})$;

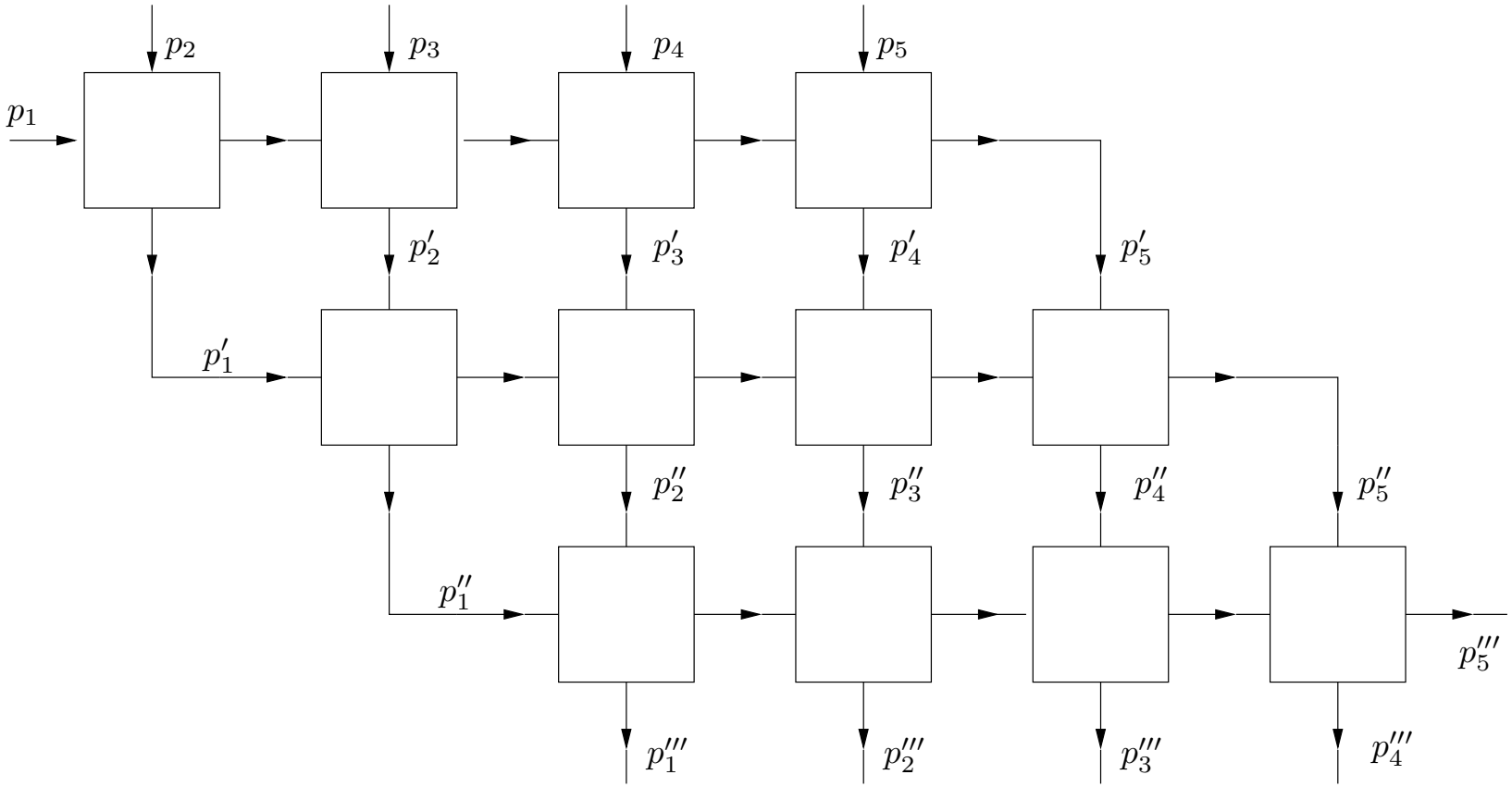
$6n$ **flops**

VecSum can be applied iteratively:

$$\sum_{i=1}^{n} p_i \xrightarrow{\textbf{VecSum}} \sum_{i=1}^{n} p'_i \xrightarrow{\textbf{VecSum}} \sum_{i=1}^{n} p''_i \xrightarrow{\textbf{VecSum}} \cdots$$

$$\Longrightarrow \quad \sum_{i=1}^{n} p_i = \sum_{i=1}^{n} p'_i = \sum_{i=1}^{n} p''_i = \cdots$$

Error-free!



Example: $n = 5$, $K = 4$.

SumK

$$\textbf{res} = \textbf{SumK}(p, K) \quad \Rightarrow \quad \textbf{res} \approx \sum_{i=1}^{n} p_i$$

function $\textbf{res} = \textbf{SumK}(p, K)$
for $k = 1 : K - 1$
  $p = \textbf{VecSum}(p);$
end
$\textbf{res} = p_n \oplus fl\left(\sum_{i=1}^{n-1} p_i\right);$ **(**$fl\,(\cdot)$**: 浮動小数点演算)**

$(6K - 5)n$ **flops**

SumKの誤差解析

$$\mathrm{cond}(\sum p_i) := \frac{\sum |p_i|}{|\sum p_i|} \quad \textbf{(総和の条件数)}$$

SumKの結果を $\mathrm{res} \in \mathbb{F}$ **とすると**

$$\frac{|\mathrm{res} - \sum p_i|}{|\sum p_i|} \leq \mathbf{u} + \mathcal{O}(\mathbf{u}^K)\mathrm{cond}(\sum p_i)$$

$\Longrightarrow$ **通常の計算精度の** K **倍**

高精度内積

TwoProductを用いると内積計算に拡張可能

For $x, y \in \mathbb{F}^n$**, using** $[h_i, r_i] =$ **TwoProduct**(x_i, y_i)

$$\sum_{i=1}^{n} x_i\, y_i = \sum_{i=1}^{n} (h_i + r_i) \qquad \left(= \sum_{i=1}^{2n} t_i \right)$$

Error-free! (provided no underflow occurs)

Dot2

function res $=$ **Dot2**(x, y)
$p = 0;\ \ s = 0;$
for $i = 1 : n$
$\quad [h, r] =$ **TwoProduct**$(x_i, y_i);$
$\quad [p, q] =$ **TwoSum**$(p, h);$
$\quad s = s \oplus (q \oplus r);$
end
res $= p \oplus s;$

$25n$ **flops**

DotK

function res $=$ **DotK**(x, y, K)
$[p, r_1] =$ **TwoProduct**(x_1, y_1);
for $i = 2 : n$
  $[h, r_i] =$ **TwoProduct**(x_i, y_i);
  $[p, r_{n+i-1}] =$ **TwoSum**(p, h);
end
$r_{2n} = p$;
res $=$ **SumK**$(r, K-1)$;

$(12K + 1)n$ **flops**

DotKの誤差解析

$x, y \in \mathbb{R}^n$. 内積の条件数は

$$\mathrm{cond}(x^T y) = \frac{2\left|x^T\right| \ |y|}{|x^T y|}.$$

DotKによる結果を $\mathsf{res} \in \mathbb{F}$ とすると

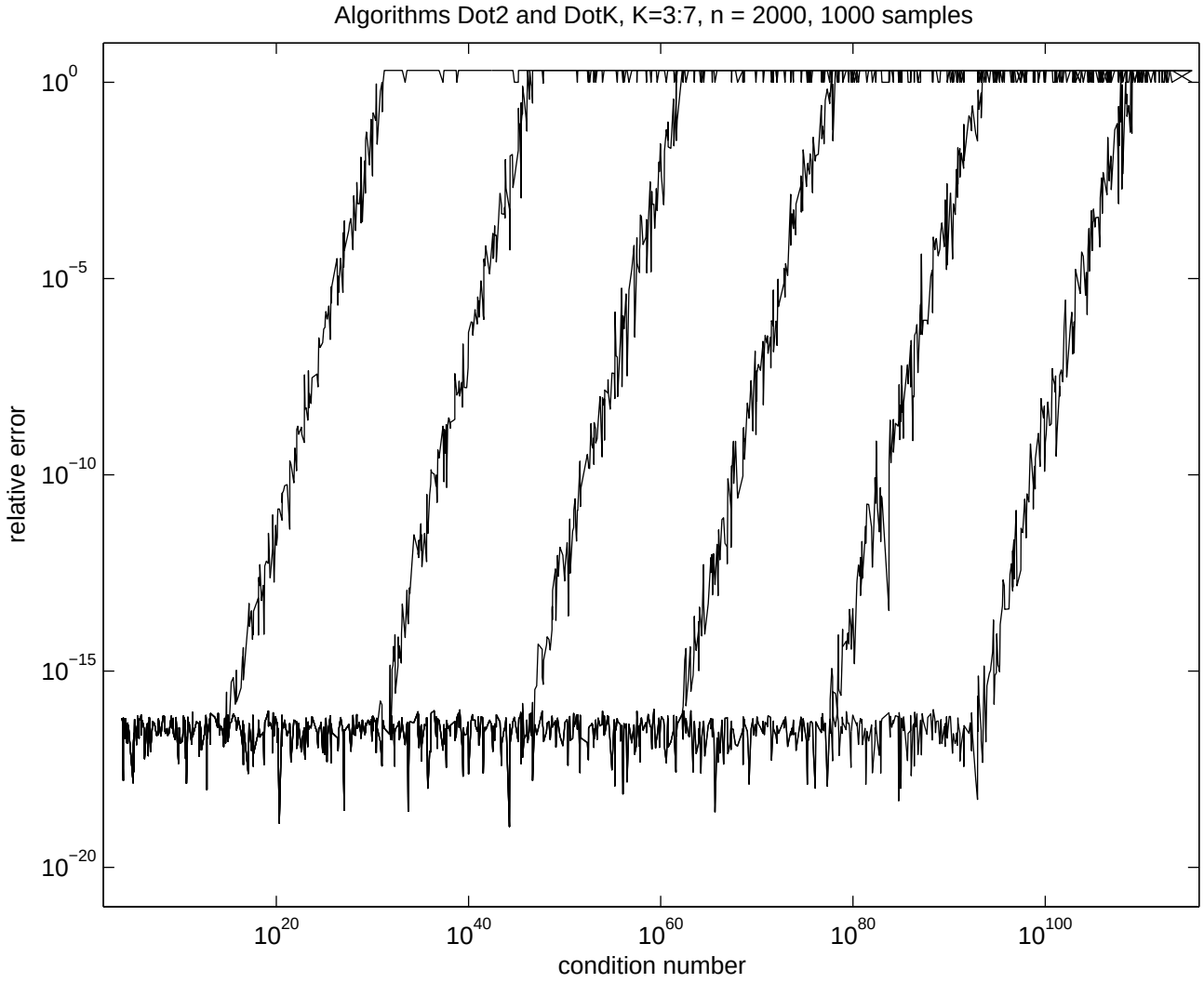
$$\frac{\left|\mathsf{res} - x^T y\right|}{|x^T y|} \leq \mathbf{u} + \mathcal{O}(\mathbf{u}^K)\mathrm{cond}(x^T y).$$

$\Longrightarrow$ 通常の計算精度の K 倍

数値実験

We evaluate accuracy of Dot2 and DotK.

- $n = 2000$ **(1000 samples)**
- **Elements of** $x, y \in \mathbb{F}^n$**: pseudo-random fl-pt numbers in** $[-1, 1]$ **s.t.** $\mathrm{cond}(x^T y)$ **is from** 2 **to** 10^{120}
- **Plot the relative error** $\dfrac{|\mathrm{res} - x^T y|}{|x^T y|}$ **where res is the result of Dot2 or DotK (**$K = 3 : 7$**)**


Algorithms Dot2 and DotK, K=3:7, n = 2000, 1000 samples
relative error
condition number
10^{0}
10^{-5}
10^{-10}
10^{-15}
10^{-20}
10^{20}
10^{40}
10^{60}
10^{80}
10^{100}

前半の結論

- 計算精度を指定できる高速なベクトルの総和・内積アルゴリズムを提案した
- 計算量だけでなく実行時間の意味でも非常に高速（単純なDD演算よりも40%程度高速，DD: double型２つで擬似4倍精度）
- 通常の浮動小数点操作しか用いていないのでポータブルかつスケーラブル

多倍長演算との違い

多倍長演算

- 計算精度（仮数部の桁数）が増える
- 取り扱える値の範囲（指数部の桁数）が増える

エラーフリー変換

- 計算精度が見かけ上増える／加減乗算には誤差がない
- 取り扱える値の範囲は変わらない

最近の話題

エラーフリー変換を応用すると

- 行列が悪条件（条件数が大きい）の場合も取り扱える
- 高精度な解を得るための方法を開発できる

後半の概要

目的 LU分解，QR分解，Cholesky分解，特異値分解，固有値分解などの行列分解のための高精度かつロバストなアルゴリズムを提案する．

A: $n \times n$実行列（複素の場合もたぶんOK）．
提案方式はAが非常に悪条件な場合も取り扱える．

$\Longrightarrow$ 例えば，IEEE 754倍精度の範囲で条件数が10^{200}を超えるような問題も取り扱える．

$\Longrightarrow$ 直接的な応用: 連立一次方程式，行列式，固有値．その他，様々な分野に応用可能．

行列の条件数

正方行列Aに対して，Aの条件数を下記のように定義する：

$$\kappa(A) := \|A\| \cdot \|A^{-1}\|.$$

$\Longrightarrow$ **$\kappa(A)$は問題の難しさの指標となる**

$\Longrightarrow$ **例えば，連立一次方程式$Ax = b$を考える（$x^* := A^{-1}b$）．**
右辺に摂動：$Ay = b + \Delta b$（$y^* := A^{-1}(b + \Delta b)$）．
このとき，解の変化量$\Delta x = y^* - x^*$は下記をみたす：

$$\frac{\|\Delta x\|}{\|x^*\|} \leq \kappa(A) \frac{\|\Delta b\|}{\|b\|}$$

IEEE 754規格の倍精度演算

$\Longrightarrow$ **1回の演算の相対精度:** $\mathbf{u} \approx 1.11 \times 10^{-16}$

$\Longrightarrow$ **条件数が**10^{16}**より大きい（**$\kappa(A) > \mathbf{u}^{-1}$**）場合，通常の倍精度演算によって得られた計算結果は1桁も合っていない場合がある．**

$\Longrightarrow$ **基本精度の限界　（基本精度：単精度や倍精度）**

$\Longrightarrow$ **このような問題を “悪条件問題” と呼ぶことにする．**

$\Longrightarrow$ **悪条件問題に対し，“結果精度の高い” 行列分解アルゴリズムを提案する．**

準備

$A = (a_{ij}), B = (b_{ij}) \in \mathbb{R}^{n \times n}$

- $|A|$：Aのすべての要素に絶対値を付けた行列，$|A| = (|a_{ij}|)$.
- $A \leq B$：すべての(i, j)に対して，$a_{ij} \leq b_{ij}$

ベクトルについても同様のnotationを利用する.

LU分解（部分軸交換付）

$A \in \mathbb{R}^{n \times n}$，$PA \approx LU$：

$$\begin{cases} P \in \mathbb{R}^{n \times n} & : \text{置換行列} \\ L \in \mathbb{R}^{n \times n} & : \text{単位下三角行列} \\ U \in \mathbb{R}^{n \times n} & : \text{上三角行列} \end{cases}$$

例）連立一次方程式$Ax = b$を解く．

$\rightarrow$ $PAx = Pb$ $\rightarrow$ $LU\widetilde{x} = b$

$\rightarrow$ $Ly = Pb$を解く． $\rightarrow$ $U\widetilde{x} = y$を解く．

LU分解の精度をどのように推定するか？

残差$\|PA - LU\|$で推定？

例えば，LU分解の残差$\|PA - LU\|$あるいは相対残差は？

$$\frac{\|PA - LU\|}{\|A\|} < \varepsilon$$

$\Longrightarrow$ **これでは情報を得られない**

例）標準的なガウスの消去法を用いた場合：

$$\|PA - LU\| \leq \mathcal{O}(\mathbf{u})\| \, |L| \, |U| \, \| \sim \mathcal{O}(\mathbf{u})\|A\|$$

メインアイディア

正確なLU分解$PA = \widehat{L}\widehat{U}$を考えると，通常，$A$の悪条件性は上三角行列$\widehat{U}$に反映される．

$$\kappa(A) \sim \kappa(\widehat{U})$$

もし，$\widehat{U}^{-1}$の良い近似X_Uが得られれば，$\kappa(A)$と比べて，$\kappa(AX_U)$は相対的に小さくなる．例えば，$\kappa(AX_U) \ll \mathbf{u}^{-1}$．

$$\|\widehat{U}X_U - I\| < 1 \tag{1}$$

$\Longrightarrow$ Aに対する前処理となる （Crout型でも同様）

AをX_Uによって前処理した後は

$$\|X_L A X_U - I\| < 1 \tag{2}$$

あるいは

$$\|A X_U X_L - I\| < 1 \tag{3}$$

を満たすような$\widehat{L}$の近似逆行列X_Lを計算するのは難しくない．

$\Longrightarrow$ 重要かつ難しい点：

どのように悪条件なAの前処理X_Uを計算するか？

仮定 $\kappa(\widehat{U}) \gg \mathbf{u}^{-1}$，$\widehat{X} := \widehat{U}^{-1}$．

$X_1 = fl\left(\widehat{X}\right)$ **とする（基本精度における最良近似）．**

$$\Longrightarrow \quad X_1 = \widehat{U}^{-1} + \Delta, \qquad |\Delta_{ij}| \sim \mathbf{u}|\widehat{U}^{-1}_{ij}| \text{（丸め誤差）}$$

このような場合でも， $\|\widehat{U}X_1 - I\| < 1$ **は必ずしも成立するわけではない：**

$$\begin{aligned} \|\widehat{U}X_1 - I\| &= \|\widehat{U}(\widehat{U}^{-1} + \Delta) - I\| = \|\widehat{U}\Delta\| \\ &\approx \|\widehat{U}\|\|\Delta\| \sim \mathbf{u} \cdot \kappa(\widehat{U}) > 1 \end{aligned}$$

$\Longrightarrow$ $\widehat{U}^{-1}$ **の近似には，なんらかの高精度な形式が必要**

前処理としての近似逆行列

丸め誤差によって，X_1 は $\widehat{U}$ の逆行列としては機能しない．しかし，$\widehat{U}$ の前処理としては機能する（$\widehat{U}$ の条件数とは無関係に）．

$$\Longrightarrow \quad \kappa(\widehat{U}X_1) = \mathcal{O}(\mathbf{u}) \cdot \kappa(\widehat{U}), \qquad \mathbf{u} \approx 10^{-16}$$

$$\Longrightarrow \quad \text{条件数を小さくできる！}$$

$$\Longrightarrow \quad \text{これを利用した積型の反復アルゴリズムを開発：}$$

$$PAX_U = \widehat{L}\underline{\underline{\widehat{U}X_1}}X_2 \ldots X_k \approx \widehat{L} \approx L$$

（反復停止条件：$\|PAX_U - L\| < \varepsilon_{\mathrm{tol}}$）

“高精度” な逆LU分解

$A \in \mathbb{R}^{n\times n}$, $PA \approx LU$:

$$\begin{cases} P \in \mathbb{R}^{n\times n} & : \text{置換行列} \\ L \in \mathbb{R}^{n\times n} & : \text{単位下三角行列} \\ U \in \mathbb{R}^{n\times n} & : \text{上三角行列} \end{cases}$$

許容誤差 $\varepsilon_{\mathrm{tol}} < 1$ に対して

$$\|L^{-1}PAU^{-1} - I\| \le \varepsilon_{\mathrm{tol}} \quad \text{あるいは} \quad \|PAU^{-1} - L\| \le \varepsilon_{\mathrm{tol}}$$

($R \approx A^{-1}$: $\|RA - I\| \le \varepsilon_{\mathrm{tol}}$ と同様のコンセプト)

提案方式の方針

悪条件問題を取り扱うためには，なんらかの高精度演算が必要．

$\Longrightarrow$ すべての計算を多倍長精度で計算するのは効率が良くない．

$\Longrightarrow$ 内積計算や行列乗算に特定すれば，比較的高速なアルゴリズムが利用可能．

$\Longrightarrow$ できる限り，高速な通常の浮動小数点演算・信頼性の高いライブラリを利用する．

高精度な行列乗算

仮定 **任意に高精度な内積の計算結果が得られる．**

$\mathbb{F}$: （基本精度における）浮動小数点数の集合
$\mathbf{u}$: 浮動小数点演算の相対精度（IEEE 754倍精度：$\mathbf{u} \approx 10^{-16}$）

できる限り，通常の浮動小数点数／浮動小数点演算を用いたい．

$\Longrightarrow$ $A_{1:p} := \sum_{i=1}^{p} A_i$，$B_{1:q} := \sum_{i=1}^{q} B_i$ のように行列を表現する．ただし，$A_i, B_i \in \mathbb{F}^{n\times n}$ かつ

$$|A_i| \leq \mathbf{u}\,|A_{i-1}|\,, \qquad |B_i| \leq \mathbf{u}\,|B_{i-1}|$$

のような関係が成り立っているものとする．

仮定 行列乗算の汎用的な関数として下記のようなものが利用可能とする：

$$C_{1:L} = \{A_{1:p}B_{1:q}\}_K^L, \quad C_{1:L} := \sum_{i=1}^{L} C_i,\ C_i \in \mathbb{F}^{n\times n}$$

は，下記をみたすとする（$c_1, c_2 : \mathcal{O}(1)$ の定数）：

$$|C_{1:\ell} - A_{1:p}B_{1:q}| \le c_1\mathbf{u}^L|A_{1:p}B_{1:q}| + c_2\mathbf{u}^K|A_{1:p}||B_{1:q}|.$$

これは，$A_{1:p}B_{1:q}$ を K 倍長精度で計算し，その結果を L 倍長精度に丸めたものに対応する（$K \ge L$ でないと意味がない）．

アルゴリズム：高精度な逆LU分解

[高精度演算が必要な箇所のみ記載．それ以外は，基本精度]

0: $X_{1:0} = I$，$k = 1$とする．

1: $B_k \leftarrow \{A \cdot X_{1:k-1}\}_k^1$． [高精度に計算 / 基本精度に丸める]

2: B_kのLU分解（$P_k B_k \approx L_k U_k$）．

3: $U_k^{-1} \approx T_k$を計算．

4: $X_{1:k} \leftarrow \{X_{1:k-1} \cdot T_k\}_k^k$． [高精度に計算 / 高精度に格納]

5: もし，$\|U_k\|\|T_k\| \leq \varepsilon_{\mathrm{tol}} \cdot \mathbf{u}^{-1}$ならば
$L := L_k$，$X_U := X_{1:k}$および$P := P_k$を返す．

6: $k \leftarrow k+1$として，1に戻る．

計算量: $\mathcal{O}(k^3 n^3)$ flops，メモリ量: $\mathcal{O}(kn^2)$ words

アルゴリズム：高精度な逆QR分解

[高精度演算が必要な箇所のみ記載．それ以外は，基本精度]

0: $X_{1:0} = I$，$k = 1$とする．

1: $B_k \leftarrow \{A \cdot X_{1:k-1}\}_k^1$． [高精度に計算 / 基本精度に丸める]

2: B_kのQR分解（$B_k \approx Q_k R_k$）．

3: $R_k^{-1} \approx T_k$を計算．

4: $X_{1:k} \leftarrow \{X_{1:k-1} \cdot T_k\}_k^k$． [高精度に計算 / 高精度に格納]

5: もし，$\|R_k\|\|T_k\| \leq \varepsilon_{\mathrm{tol}} \cdot \mathbf{u}^{-1}$ならば
$Q := Q_k$，$X_R := X_{1:k}$を返す．

6: $k \leftarrow k+1$として，1に戻る．

難しい点

- アルゴリズムは簡潔だが，解析が難しい
- 実際，厳密な解析は不可能（期待値等，確率的な話になる）
- ある程度の自然な仮定の下で，解析可能

Proposition 1. $A \in \mathbb{F}^{n\times n}$ **とする．** $X_{1:k}$ **を逆** LU **分解（逆** QR **分解）アルゴリズムによって得られた** $n \times n$ **上三角行列とする．このとき，ある仮定の下で，** $k \geq 1$ **に対して下記が成立する：**

$$\kappa(AX_{1:k}) = 1 + \mathcal{O}(\mathbf{u}^k) \cdot \kappa(A) \tag{4}$$

Remark 1. $A = \sum_{i=1}^{m} A_i$, $A_i \in \mathbb{F}^{n\times n}$ **のような入力（多倍長精度の入力に相当）でも取り扱えるようにアルゴリズムを拡張するのは難しくない（もし，高精度な内積計算／行列乗算のアルゴリズムが利用可能なら）．**

$\|\widehat{U}X_{1:k} - I\| \approx \|L_k^{-1}PAX_{1:k} - I\| < 1$ **を達成するためには，下記のような** k **で十分．**

$$k = \left\lceil \frac{\log[\varepsilon_{\mathrm{tol}} \cdot \kappa(A)^{-1}]}{\log \mathbf{u}} \right\rceil .$$

しかしながら，一般的に $\kappa(A)$ **は事前にはわからないため，** k **をあらかじめ決めることはできない．**

そこで，下記のような停止条件を利用：

$$\|U_k\|\|T_k\| < \varepsilon_{\mathrm{tol}} \cdot \mathbf{u}^{-1} \tag{5}$$

(5) が成立 $\Longrightarrow$ $\|L_k^{-1}PAX_{1:k} - I\| < \varepsilon_{\mathrm{tol}}$

提案アルゴリズムの特徴

1. 問題の難しさに応じて，必要なだけ反復的に計算精度を増やすことができる（アダプティブ性）．
2. 高精度演算は，行列乗算（つまり内積計算）のみ．
3. それ以外の処理は，BLAS，LAPACKなどを利用可能で，すべて通常の浮動小数点演算で実行（高速性）．
4. 高精度な内積計算／行列乗算が高速化されると，提案アルゴリズムも高速化される．

2についても，Level-3 BLASを利用した高速な方式を開発中（早大・尾崎氏との共同研究）．

数値実験

逆LU分解／逆QR分解のふるまいについて数値実験する．

- **倍精度を基本精度とする（$\mathbf{u} = 2^{-53} \approx 1.1 \cdot 10^{-16}$）**
- **テスト行列：Rumpの行列（INTLABの`randmat(n,cnd)`）**
- $\mathtt{n} = 100$，$\mathtt{cnd} = 10^{100}$（$A \in \mathbb{F}^{100\times 100}$，$\kappa(A) \approx 1.75 \cdot 10^{107}$）

Table 1: Rumpの行列（$n = 100$ and $\kappa(A) \approx 1.75 \cdot 10^{107}$）に対する逆LU分解の結果

k	$\kappa(U_k)$	$\kappa(T_k)$	$\kappa(AX_{1:k})$	$\mathbf{u}^k\kappa(A)$
1	$3.50 \cdot 10^{18}$	$3.50 \cdot 10^{18}$	$2.37 \cdot 10^{93}$	$1.94 \cdot 10^{91}$
2	$5.28 \cdot 10^{18}$	$5.28 \cdot 10^{18}$	$2.18 \cdot 10^{78}$	$2.16 \cdot 10^{75}$
3	$4.01 \cdot 10^{18}$	$4.01 \cdot 10^{18}$	$1.79 \cdot 10^{64}$	$2.39 \cdot 10^{59}$
4	$4.85 \cdot 10^{18}$	$4.85 \cdot 10^{18}$	$3.45 \cdot 10^{48}$	$2.66 \cdot 10^{43}$
5	$1.99 \cdot 10^{18}$	$1.99 \cdot 10^{18}$	$6.77 \cdot 10^{33}$	$2.95 \cdot 10^{27}$
6	$1.16 \cdot 10^{18}$	$1.16 \cdot 10^{18}$	$1.30 \cdot 10^{18}$	$3.27 \cdot 10^{11}$
7	$2.73 \cdot 10^{17}$	$2.73 \cdot 10^{17}$	$3.96 \cdot 10^{2}$	< 1
8	$1.91 \cdot 10^{2}$	$1.91 \cdot 10^{2}$	$8.68 \cdot 10^{1}$	< 1

Table 2: Rumpの行列（$n = 100$ and $\kappa(A) \approx 1.75 \cdot 10^{107}$）に対する逆QR分解の結果

k	$\kappa(R_k)$	$\kappa(T_k)$	$\kappa(AX_{1:k})$	$\mathbf{u}^k\kappa(A)$
1	$3.27 \cdot 10^{19}$	$3.27 \cdot 10^{19}$	$2.00 \cdot 10^{93}$	$1.94 \cdot 10^{91}$
2	$1.86 \cdot 10^{19}$	$1.86 \cdot 10^{19}$	$1.06 \cdot 10^{77}$	$2.16 \cdot 10^{75}$
3	$7.97 \cdot 10^{17}$	$7.97 \cdot 10^{17}$	$1.61 \cdot 10^{62}$	$2.39 \cdot 10^{59}$
4	$2.20 \cdot 10^{17}$	$2.20 \cdot 10^{17}$	$4.23 \cdot 10^{46}$	$2.66 \cdot 10^{43}$
5	$2.31 \cdot 10^{17}$	$2.31 \cdot 10^{17}$	$2.00 \cdot 10^{32}$	$2.95 \cdot 10^{27}$
6	$4.04 \cdot 10^{17}$	$4.04 \cdot 10^{17}$	$6.69 \cdot 10^{16}$	$3.27 \cdot 10^{11}$
7	$2.18 \cdot 10^{18}$	$2.18 \cdot 10^{18}$	$1.39 \cdot 10^{3}$	< 1
8	$1.39 \cdot 10^{3}$	$1.39 \cdot 10^{3}$	$1.00 \cdot 10^{0}$	< 1

応用(1): 連立一次方程式

連立一次方程式の近似解の計算に提案アルゴリズムを適用：

1. A^T **の逆LU分解（Crout型なら** A **でよい）**

$$PA^T X_U \approx L \quad \Leftrightarrow \quad X_U^T AP \approx L^T$$

2. $\widetilde{y} = \{X_U^T \cdot b\}_m^1$ **を計算**

3. **三角方程式** $L^T z = \widetilde{y}$ **を解く（得られた近似解を** $\widetilde{z}$ **とする）**

4. $\widetilde{x} = P\widetilde{z}$ **を計算**

数値実験(1): (スケール化)Hilbert行列 H_n

H_n は要素が整数の行列 ($n \leq 21$ のとき，倍精度で正確に表現可能)

- $n = 15$ $(\kappa(H_{15}) \approx 6.12 \times 10^{20})$
- 右辺：$b = H_{15}e \in \mathbb{F}^{15}$，$e := (1, \ldots, 1)^T$
- $H_{15}x = b$ の真の解：$H_{15}^{-1}b = e = (1, \ldots, 1)^T$

(Matlab demo)

数値実験(2): Rumpの行列

様々な条件数に対し，提案アルゴリズムを用いて得られた連立一次方程式の近似解の相対精度（ノルム評価）をみる．

- Rumpの行列 `randmat`（$n = 500$，条件数は10^{20}から10^{100}）
- $b = (1, \ldots, 1)^T$
- 提案アルゴリズムの許容誤差：$\varepsilon_{\mathrm{tol}} = 10^{-6}$
- 多倍長精度との速度比較のため，GMPベースのガウスの消去法も実行（trial-and-error方式）

```
function x = test_gmp_lin(A,b,tol)
d = 53;   % equal to using double precision
xt = A\b; % Solve Ax = b in double precision
while 1
    d = 2*d;                % d = 106, 212, 424, ...
    % Solve Ax = b by GMP-based GEPP
    x = gmp_lin(A,b,d);
    if norm(xt-x) <= tol*norm(x), break, end
    xt = x;
end
```

Table 3: Results for Rump's matrices with $n = 500$

	Proposed algorithm			GMP-based GEPP		
$\kappa(A)$	ε_1	t_1	m	ε_2	t_2	$d/53$
$1.3 \cdot 10^{20}$	$2.9 \cdot 10^{-13}$	1.28	2	$1.2 \cdot 10^{-16}$	52.02	4
$1.7 \cdot 10^{33}$	$2.5 \cdot 10^{-15}$	3.95	3	$1.2 \cdot 10^{-16}$	125.24	8
$2.7 \cdot 10^{43}$	$4.2 \cdot 10^{-15}$	8.87	4	$1.2 \cdot 10^{-16}$	125.24	8
$1.1 \cdot 10^{49}$	$1.2 \cdot 10^{-13}$	9.23	4	$1.2 \cdot 10^{-16}$	125.79	8
$5.0 \cdot 10^{59}$	$3.1 \cdot 10^{-15}$	16.27	5	$1.2 \cdot 10^{-16}$	125.92	8
$7.0 \cdot 10^{73}$	$5.1 \cdot 10^{-15}$	25.94	6	$1.2 \cdot 10^{-16}$	304.96	16
$3.9 \cdot 10^{81}$	$7.6 \cdot 10^{-11}$	26.39	6	$1.2 \cdot 10^{-16}$	315.46	16
$2.7 \cdot 10^{92}$	$5.4 \cdot 10^{-13}$	39.88	7	$1.2 \cdot 10^{-16}$	311.71	16
$8.0 \cdot 10^{103}$	$4.2 \cdot 10^{-15}$	56.44	8	$1.2 \cdot 10^{-16}$	317.07	16

応用例(2): 行列式の計算

良い方法として知られているのは，LU分解してU_{ii}の積を計算：

$$\det(A) \approx \det(P^T LU) = \det(P)\det(U).$$

（もちろん，これは近似計算なので，結果の保証がない）

行列式は，その符号判定の情報が重要な例が多い．

行列式の精度保証

X_L, X_U: L, U の近似逆行列.

$$B := X_L P A X_U \Rightarrow \det(B) = \det(P)\det(A)\det(X_U) \Rightarrow$$

$$\det(A) = \det(P)\ \det(B)/\det(X_U) \tag{6}$$

$\Longrightarrow$ B: 単位行列（対角行列）に近い, $\det(B) \approx 1$

$\Longrightarrow$ Gershgorinの定理により，Bの固有値の範囲がわかる

$\Longrightarrow$ $\det(B)$の包含が可能

$\Longrightarrow$ (6)により，$\det(A)$の包含も可能

しかしながら，Aが**悪条件**な場合，LもUも良いLU分解因子とならない．

$\Longrightarrow$ その場合，$B = X_L P A X_U$は単位行列（対角行列）とかけ離れてしまう．

$\Longrightarrow$ Gershgorinの定理が意味をなさない．

この問題も，提案アルゴリズムで解決できる！

(Matlab demo)

おわりに

今回のコンセプトを用いると，高精度かつロバストな

- 逆Cholesky分解（先日，EASIAMで発表）
- 逆$\mathrm{LDL}^{\mathrm{T}}$分解
- 特異値分解（先日，JSIAM年会で発表）
- 固有値分解（対称行列）（今度，計算工学会で発表）

も構成できる（多少の修正は必要．解析も別）．

ありがとうございました.