精度保証付き数値計算の基礎
2章：丸め誤差解析と高精度計算

尾崎 克久

芝浦工業大学システム理工学部数理科学科
共同執筆者：荻田 武史（東京女子大学）

チュートリアル，2018年9月10日，早稲田大学

内容

第2章の内容：

- ベクトルの総和，内積に関する誤差解析
- エラーフリー変換
- 高精度計算（総和・内積）

ここでも証明は省略する

数値計算

$p \in \mathbb{F}^n$ に対して，$\sum_{i=1}^{n} p_i$ を浮動小数点演算を用いて計算．

この計算では最大で $n-1$ 回の誤差が入る可能性があり，その最大の影響を調べる．

絶対誤差の上限についての誤差評価を解説していく．

数値計算

総和を考える際には，一般に計算の順序を考える必要．

C言語で配列aの要素の総和を計算する．

例えば

$$\texttt{for}\ (i = 1,\ r = a[0];\ i < n;\ i++)\ r = r + a[i];$$

とコードを作成し，コンパイルを行ったとする．

数値計算

近年のコンパイラでは

$$(((a[0] + a[1]) + a[2]) + a[3] \ldots)$$

と計算をするコードが生成されている保証はない．

よって，任意の計算順序に対して成立する誤差評価を与えることが汎用性を持つ．

「浮動小数点演算の結果の精度に影響する最適化をしない」というコンパイルオプションを使用した場合には別である．

デモ

$$a_1 = 1, \quad a_2 = a_3 = \cdots = a_n = \mathbf{u}$$

とおいて，この総和を考える．

$$\mathrm{fl}(\ (((a_1 + a_2) + a_3) + \ldots) + a_n\ )$$

と逐次計算しているならば，計算結果は1である．

しかし ····

数値計算

デモ

数値計算

ここで，任意の順序による浮動小数点演算の結果を表す表記float(·)を導入する．

float($a + b + c + d$)は

$$\mathrm{fl}(((a + b) + c) + d),\ \mathrm{fl}(((a + c) + b) + d),\ \mathrm{fl}(((a + c) + d) + b)$$

$$\mathrm{fl}((a + b) + (c + d)),\ \mathrm{fl}((a + c) + (c + d)),\ \mathrm{fl}((a + d) + (b + c))$$

などのどれでも良いことを意味する．

数値計算

$$\mathrm{float}(a + b + c) < 1$$

ならば

$$\mathrm{fl}((a + b) + c) < 1, \quad \mathrm{fl}(a + (b + c)) < 1, \quad \mathrm{fl}((a + c) + b) < 1$$

のすべてを満たす.

2分木

ある式に対する演算の過程は2分木によって整理できるため，2分木の構造を利用した帰納法を用いる．次ページの図は，

$$((p_1 + p_7) + (p_6 + p_2)) + ((p_3 + p_5) + (p_4 + p_8))$$

という演算順序に対する2分木である．

2分木

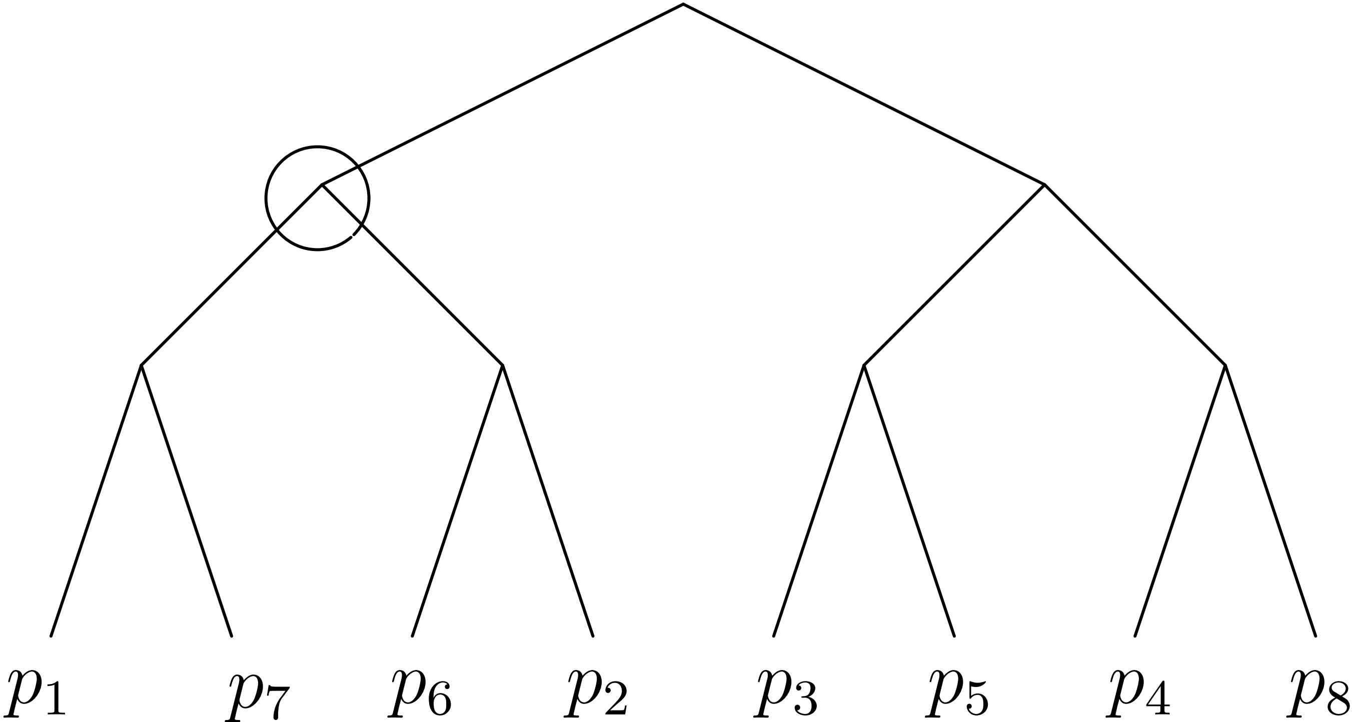

Figure 1: 2分木を利用した演算順序

丸め誤差解析

まずは，2分木の葉について証明を行う.

次に，あるノードに対する左側の子と右側の子に対し，それぞれにある誤差評価が成立したと仮定する.

そのノードでの誤差評価が成立するかを検証し，帰納的に証明する手法である.

誤差の上限

定理 1 $p \in \mathbb{F}^n$ に対する $\sum_{i=1}^n p_i$ を考える．任意の計算順序によって得られた計算値 $\mathrm{float}(\sum_{i=1}^n p_i)$ に対して，誤差の上限を表す以下の不等式が成立する．

$$\left|\mathrm{float}\left(\sum_{i=1}^n p_i\right) - \sum_{i=1}^n p_i\right| \le (n-1)\mathbf{u} \sum_{i=1}^n |p_i| \tag{1}$$

誤差の上限

歴史的な背景を言えば，定理1における$(n-1)\mathbf{u}$の代わりに，$(1+\mathbf{u})^{n-1}-1$，あるいは$(n-1)\mathbf{u}<1$という条件下で

$$\gamma_n := \frac{(n-1)\mathbf{u}}{1-(n-1)\mathbf{u}}$$

が使用されていた．$(n-1)\mathbf{u}$という簡単な係数を使用できるのはSiegfried M. Rump氏やClaude-Pierre Jeannerod氏らの研究成果の賜物である．

誤差の上限

定理 2 $p \in \mathbb{F}^n$ に対して

$$|\mathrm{float}(\sum_{i=1}^{n} p_i) - \sum_{i=1}^{n} p_i| \le (n-1)\mathbf{u} \cdot \mathrm{ufp}(\mathrm{float}(\sum_{i=1}^{n} |p_i|)) \quad (2)$$

が成立する．ただし，float· 内の計算順序は不等式の左辺と右辺で同じでなければならない．

この丸め誤差評価は最適である

誤差の上限

$(n-1)\mathbf{u} \le 1$ のとき，$\mathbf{u}$ と $\mathrm{ufp}(\cdot)$ はともに2のべき乗数であるから

$$\left|\mathrm{float}\left(\sum_{i=1}^{n} p_i\right) - \sum_{i=1}^{n} p_i\right| \le \mathrm{float}\left((n-1)\mathbf{u} \cdot \mathrm{ufp}\left(\mathrm{float}\left(\sum_{i=1}^{n} |p_i|\right)\right)\right)$$

と誤差の上限をすべて浮動小数点演算で評価できる．

内積の誤差評価

アンダーフローが発生した場合は$\mathbf{u}_S$の定数倍を考慮すれば良い.

これ以降ではアンダーフローが発生しないと仮定して議論を進める.

定理 3 $x, y \in \mathbb{F}^n$に対して

$$|\mathrm{float}\, x^T y - x^T y| \le n\mathbf{u}|x^T||y| \tag{3}$$

が成立する.

内積の誤差評価

定理 4 $x, y \in \mathbb{F}^n$ に対して，$(n-1)\mathbf{u} \leq 1$ のとき以下の不等式が成立する．

$$|\mathrm{float}\, x^T y - x^T y| \leq (n+2)\mathbf{u} \cdot \mathtt{ufp}(\mathrm{float}|x^T||y|) \qquad (4)$$

ただし，float· 内の計算順序は不等式の左辺と右辺で同じでなければならない．

ある範囲内であれば浮動小数点演算で上限が求まる．

無誤差変換

IEEE 754規格が定める2つの浮動小数点数がある．

浮動小数点演算の結果を2つの浮動小数点数の和
に変換する方法は高精度計算に非常に有用である．

MATLAB関数のような入力と出力の形式で関数を記述．

無誤差変換

$a, b \in \mathbb{F}$に対する$\mathrm{fl}(a+b)$は丸め誤差のために真値$a+b$と一致するとは限らないことは先に紹介した.

$x = \mathrm{fl}(a+b)$とすると，$y = a+b-x$となる$y \in \mathbb{F}$を浮動小数点演算で求めるアルゴリズムを紹介する.

Dekkerは$|a| \ge |b|$が成立するとき，$a+b = x+y$と変換するアルゴリズムを提案した.

これにより，浮動小数点演算による和の結果において，誤差により失う情報を保持できる.

定理 5 (Dekker) $a, b \in \mathbb{F}$について，$|a| \geq |b|$のとき，以下の関数FastTwoSumを実行すれば

$$a + b = x + y,\ |y| \leq \mathbf{u} \cdot \mathtt{ufp}(a + b) \tag{5}$$

が成立する．

function $[x, y] = \mathtt{FastTwoSum}(a, b)$
　$x = \mathrm{fl}(a + b);$
　$y = \mathrm{fl}(b - (x - a));$
end　□

無誤差変換

例えば$a = 1,\ b = 2^{100}$に対してFastTwoSumを実行すると$x = 2^{100}, y = 0$となる

$a + b = x + y$が成立しない

ただし，必ずしも$|a| \geq |b|$でなくともよい．

$a \in 2\mathbf{u} \cdot \mathtt{ufp}(b)\mathbb{Z}$であれば$a + b = x + y$が成立する．

a, bの大小関係に何も仮定を置かなければ，下記のKnuthによるアルゴリズムにより$a + b = x + y$と変換される．

定理 6 (Knuth) $a, b \in \mathbb{F}$について，以下の関数`TwoSum`を実行すれば，定理*5*で示した式が成立する．

$$
\begin{aligned}
&\textbf{function} \quad [x, y] = \mathtt{TwoSum}(a, b)\\
&\qquad x = \mathrm{fl}(a + b);\\
&\qquad z = \mathrm{fl}(x - a);\\
&\qquad y = \mathrm{fl}((a - (x - z)) + (b - z));\\
&\mathrm{end}
\end{aligned}
$$

□

オーバーフローが起きる場合を除けば，$[x, y] = \mathtt{TwoSum}(a, b)$ の結果は，下記のアルゴリズムの結果と等価である．

$$
\begin{aligned}
&\mathtt{if}\ |a| \ge |b| \\
&\quad [x, y] = \mathtt{FastTwoSum}(a, b); \\
&\mathtt{else} \\
&\quad [x, y] = \mathtt{FastTwoSum}(b, a); \\
&\mathtt{end}
\end{aligned}
$$

ただし，上記の計算法より，分岐処理のない定理6のアルゴリズムが計算速度の点で有利になる場合がある．

Fused Multiply-Add

$a, b, c \in \mathbb{F}$に対する$a * b + c$の計算において，通常の浮動小数点演算では$a * b$の評価に誤差が発生し，その結果とcの和にも誤差が発生する可能性がある．

最大で2回の誤差が発生する

IEEE 754-2008規格には，このFMAが仕様として定められている．

Fused Multiply-Add

最大で2回の誤差が発生する$a * b + c$の評価を実数演算で計算し，その結果を浮動小数点数に丸めた結果を出力できるFMA (Fused Multiply-Add) という機能がある．

近年のCPUやGPGPUなどで利用できることが多い．

例：Intel Haswell以降（第4世代）

Fused Multiply-Add

FMA を用いて $a * b + c$ を計算する表記を $\mathtt{FMA}(a, b, c)$ とする.

FMA を使用すれば，$a * b - \mathrm{fl}(a * b)$ を簡単に求められる.

（FMA を用いなくても可能ではある）

ここでも浮動小数点演算においてアンダーフローが起こらないことを仮定する.

定理 7 $a, b \in \mathbb{F}$について，以下の関数`TwoProdFMA`により，$a * b = x + y$が成立する．

$$
\begin{aligned}
&\textbf{function} \quad [x, y] = \texttt{TwoProdFMA}(a, b) \\
&\quad x = \mathrm{fl}(a * b); \\
&\quad y = \texttt{FMA}(a, b, -x); \\
&\mathrm{end}
\end{aligned}
$$

□

定理7においてxはabの計算値であり，$ab - x$を実数計算した後に最も近い浮動小数点数に丸めることは，$\mathrm{fl}(ab) - ab$を計算していることになる．

定理 8 (Ogita-Rump-Oishi) $p \in \mathbb{F}^n$

$$\sum_{i=1}^{n} p_i = \sum_{i=1}^{n} p'_i, \quad \sum_{i=1}^{n-1} |p'_i| \le \gamma_{n-1} \sum_{i=1}^{n} |p_i|$$

function $p' = \texttt{VecSum}(p)$

$p' = p;$

$\texttt{for}\ i = 2 : n$

$[p'_i, p'_{i-1}] = \texttt{TwoSum}(p'_i, p'_{i-1});$

end

end □

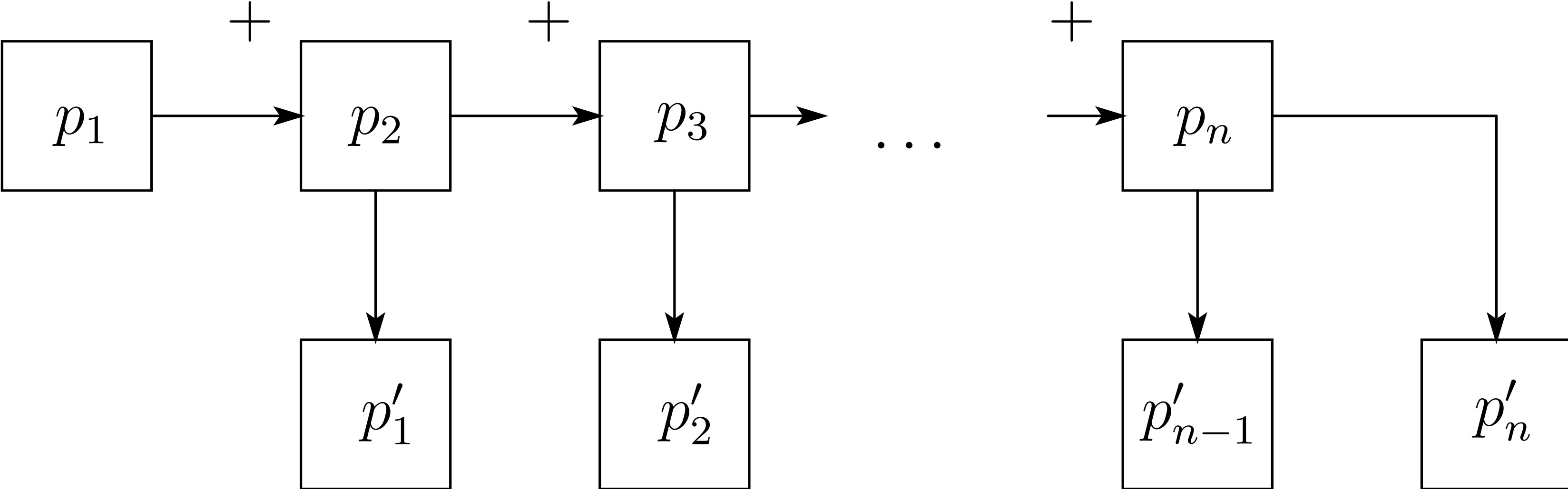

Figure 2: `VecSum`のイメージ

説明

関数VecSumを実行すると$p'_n = \mathrm{fl}(\sum_{i=1}^{n} p_i)$である[1].

通常の浮動小数点演算で総和を計算したとき，誤差によって失われてしまう情報を$p'_i\ (1 \le i \le n-1)$により保存.

pとp'は「ベクトルの総和」として情報は等価.

$p'_i\ (1 \le i \le n-1)$を再利用すれば，高精度計算アルゴリズムの開発が可能.

[1] $\mathrm{fl}\sum_{i=1}^{n} p_i$は$\mathrm{fl}(\cdots \mathrm{fl}(\mathrm{fl}(p_1 + p_2) + p_3) + \cdots)$と，前から順に浮動小数点演算で評価する意味とする.

定理 9 (Ogita-Rump-Oishi) $p \in \mathbb{F}^n$ に対して，以下のアルゴリズムを $K = 2$ として実行したとき

$$\left| z - \sum_{i=1}^{n} p_i \right| \le \mathbf{u} \left| \sum_{i=1}^{n} p_i \right| + \gamma_{n-1}^2 \sum_{i=1}^{n} |p_i| \tag{6}$$

が成立する．また $K \ge 3$ のとき，$4(n-1)\mathbf{u} \le 1$ ならば

$$\left| z - \sum_{i=1}^{n} p_i \right| \le (\mathbf{u} + 3\gamma_{n-1}^2) \left| \sum_{i=1}^{n} p_i \right| + \gamma_{2n-2}^K \sum_{i=1}^{n} |p_i| \tag{7}$$

が成立する．

```
function    z = SumK(p, K)
    for  k = 1 : K − 1
        p^(k) = VecSum(p^(k−1));
    end
    z = fl(p_n^(K−1) + Σ_{i=1}^{n−1} p_i^(K−1));
end
```

$$\textbf{function} \quad z = \mathtt{SumK}(p, K)$$
$$\text{for } k = 1 : K - 1$$
$$p^{(k)} = \mathtt{VecSum}(p^{(k-1)});$$
$$\text{end}$$
$$z = \mathrm{fl}\left(p_n^{(K-1)} + \sum_{i=1}^{n-1} p_i^{(K-1)}\right);$$
$$\text{end} \qquad \square$$

高精度計算

通常の浮動小数点演算でベクトルの総和を計算した場合の定理1に対して，(6)では$\sum_{i=1}^{n} |p_i|$の係数が$\mathbf{u}^2$のオーダーで小さくなる．

また(7)では$\sum_{i=1}^{n} |p_i|$の係数が$\mathbf{u}^K$のオーダーで小さくなることが，このアルゴリズムの特徴を表している．

さらに高精度計算の結果とともに，誤差の上限を浮動小数点数として出力するアルゴリズムもある．

なぜ総和が重要か？

例えば$a, b, c \in \mathbb{F}$に対して

$$abc = (d + e)c = dc + ec = f + g + h + i$$

のように，足し算に誤差なく変換される．

（$d, e, f, g, h, i \in \mathbb{F}$）

内積，多項式，行列式なども総和に変形できる．

まとめ

本チュートリアルでは

- 丸め誤差解析
- エラーフリー変換
- 高精度計算

について解説した．
詳細は本に記載してあります．

もし，FMAが利用できない環境においても`TwoProdFMA`と同じ結果を得るアルゴリズムを紹介する.

定理 10 (Veltkamp) $a, b \in \mathbb{F}$について，以下の関数`TwoProd`により，$a * b = x + y$が成立する.

$$
\begin{aligned}
&\textbf{function} \quad [x, y] = \texttt{TwoProd}(a, b) \\
&\quad [a_h, a_l] = \texttt{Split}(a); \\
&\quad [b_h, b_l] = \texttt{Split}(b); \\
&\quad x = \mathrm{fl}(a * b); \\
&\quad y = \mathrm{fl}(a_l * b_l - (((x - a_h * b_h) - a_l * b_h) - a_h * b_l)); \\
&\texttt{end}
\end{aligned}
$$

□

上記の`TwoProd`(a, b)に使用されている関数`Split`を具体的に紹介する.

定理 11 (Dekker) $a, b \in \mathbb{F}$について，以下の関数`Split`により$a = x + y$が成立する.

$$
\begin{aligned}
&\textbf{function} \quad [x, y] = \texttt{Split}(a) \\
&\quad c = \mathrm{fl}(\texttt{factor} * a); \quad \% \ \texttt{factor} = 2^{\lceil (-\log_2 \mathbf{u})/2 \rceil} + 1 \\
&\quad x = \mathrm{fl}(c - (c - a)); \\
&\quad y = \mathrm{fl}(a - x); \\
&\texttt{end} \qquad \qquad \square
\end{aligned}
$$

定理 12 (Rump-Ogita-Oishi) $a, \sigma \in \mathbb{F}$ ($\sigma = 2^k \geq |a|$, $k \in \mathbb{Z}$)について，以下の関数`ExtractScalar`を実行すると$a = x + y$，$x \in \mathbf{u}\sigma\mathbb{Z}$, $|y| \leq \mathbf{u}\sigma$を満たす.

function $[x, y] =$ `ExtractScalar`(a, σ)

$x = \mathrm{fl}((a + \sigma) - \sigma)$;

$y = \mathrm{fl}(a - x)$;

end □

$$\sigma \geq 2^M a,\ M \in \mathbb{N} \ \Rightarrow\ |x| \leq 2^{-M}\sigma \tag{8}$$

定理 13 (Rump-Ogita-Oishi) $p \in \mathbb{F}^n$について，$\sigma = 2^k \in \mathbb{F}, k \in \mathbb{Z}$が

$$\sigma \geq 2^{\lceil \log_2 n \rceil} \cdot 2^{\lceil \log_2 \max_{1 \leq i \leq n} |p_i| \rceil}$$

を満たすとする．

次の関数`ExtractVector`を実行したときに，

$$\sum_{i=1}^{n} p_i = \tau + \sum_{i=1}^{n} p'_i \tag{9}$$

が成立する．

function $[\tau, p'] = \texttt{ExtractVector}(p, \sigma)$

$\tau = 0;$

for $i = 1 : n$

$[t_i, p'_i] = \texttt{ExtractScalar}(p_i, \sigma);$

$\tau = \mathrm{fl}(\tau + t_i);$

end

end □

定理 14 (Rump-Ogita-Oishi) $p \in \mathbb{F}^n$ に対して，以下のアルゴリズムを実行すると

$$\sum_{i=1}^{n} p_i = \tau_1 + \tau_2 + \sum_{i=1}^{n} q_i, \quad q \in \mathbb{F}^n,\ \tau_1, \tau_2 \in \mathbb{F}$$

が成立する．

```
function [τ1, τ2, q, σ] = Transform(p)
  q = p; t' = 0; μ = max(|p_i|);
  M = NextPowerTwo(n + 2); σ' = M * NextPowerTwo(μ);
  repeat
    t = t'; σ = σ';
    [τ, q] = ExtractVector(σ, q);
    t' = fl(t + τ);
    if t' = 0, [τ1, τ2, q, σ] = Transform(q); return; end
    σ' = fl(M * u * σ);
  until |t'| ≥ fl(M^2 * u * σ) or σ ≤ 1/2 * u^-1 * u_S
  [τ1, τ2] = FastTwoSum(t, τ);
```

- 上記にある$0 \neq a \in \mathbb{F}$に対する`NextPowerTwo`(a)は，a以上である最小の2のべき乗数，すなわち$2^{\lceil \log_2 |a| \rceil}$を返す関数とする．

- MATLABの関数`nextpow2`(a)は，指数の部分$\lceil \log_2 |a| \rceil$を返す．

- τ_1については，$\tau_1 + \tau_2 + \sum_{i=1}^{n} q_i$を計算しても桁落ちが発生しない性質を持つ．

定理 15 (Rump-Ogita-Oishi) $p \in \mathbb{F}^n$について，次のアルゴリズムを実行すると

$$\begin{cases} \sum_{i=1}^n p_i \in \mathbb{F} & \Longrightarrow \quad \sum_{i=1}^n p_i = z \\ \sum_{i=1}^n p_i \notin \mathbb{F} & \Longrightarrow \quad \left|\sum_{i=1}^n p_i - z\right| \le \mathbf{u} \cdot \mathtt{ufp}(z) \end{cases}$$

を満たす．

function $z = \mathtt{AccSum}(p)$
　$[\tau_1, \tau_2, p'] = \mathtt{Transform}(p)$;
　$z = \mathrm{fl}\,\tau_1 + (\tau_2 + (\sum p'))$;
end □

定理 16 (Ogita-Rump-Oishi) $x, y \in \mathbb{F}^n$ について，以下の関数`DotK`を実行すると$K \geq 2$について以下が成立する．

$$|z - x^T y| \leq (\mathbf{u} + 2\gamma_{4n-2}^2)|x^T y| + \gamma_{4n-2}^K |x^T||y|$$

```
function z = DotK(x, y, K)
  [p, r_1] = TwoProd(x_1, y_1); または [p, r_1] = TwoProdFMA(x_1, y_1);
  for i = 2 : n
    [h, r_i] = TwoProd(x_i, y_i); または [h, r_i] = TwoProdFMA(x_i, y_i);
    [p, r_{n+i-1}] = TwoSum(p, h);
  end
  r_{2n} = p;
  z = SumK(r, K - 1);
end
```

□

Faithful Rounding

誤差評価式(10)から，定理15による結果はfaithful roundingと言われる．また

$$\left| \sum_{i=1}^{n} p_i - z \right| \leq \frac{1}{2}\mathbf{u} \cdot \mathtt{ufp}(z) \tag{10}$$

となる$z \in \mathbb{F}$を出力するアルゴリズムも提案されている．(6)や(7)の結果は，$|\sum_{i=1}^{n} p_i|$の値が小さく，$\sum_{i=1}^{n} |p_i|$の値が大きければ，相対誤差は大きくなるが，(10)では問題に依存しないことがわかる．