お話：数値解析 第9回
丸め誤差

長田直樹

1　はじめに

数値計算は、無限にある実数を有限個の数 — 手計算の場合は有限桁の数、計算機による数値計算の場合は有限個の浮動小数点数または固定小数点数 — を用いて計算するため、丸め誤差を伴う。

丸め誤差の知識は、数値計算結果の評価に不可欠であり、桁落ちや情報落ちなどの回避にも役立つ。今回は数値計算における丸め誤差の話をする。

2　固定小数点数

実数を小数で表す際に固定小数点数と浮動小数点数が用いられる。円周率 3.14 は 10 進固定小数点数で、アボガドロ定数 6.02×10^{23} は 10 進浮動小数点数である。計算機で実数を扱うときにも固定小数点数と浮動小数点数の 2 つの方式がある。

整数部と小数部の桁数が決まった表示方法が**固定小数点数**である。

整数部 q 桁と小数部 r 桁の 10 進固定小数点数は全部で $2 \times 10^{q+r} - 1$ 個 (0 以外は正負 2 つ) ある。

以下では何進数かを明示するときは丸括弧をつけた下付きの添字を用いる。2 進数は 0 と 1 により表され、0 と 1 の個数はビットという単位で数える。2 進整数と 2 進小数があり

$$13_{(10)} = 2^3 + 2^2 + 2^0 = 1101_{(2)},$$
$$3.25_{(10)} = 2^1 + 2^0 + 2^{-2} = 11.01_{(2)}$$

というように表す。$1101_{(2)}$ は 4 ビット 2 進整数、$11.01_{(2)}$ は整数部 2 ビット小数部 2 ビットの 2 進固定小数点数である。

整数部 q ビットと小数部 r ビットの 2 進固定小数点数は全部で $2^{1+q+r} - 1$ 個ある。

COBOL などのプログラミング言語、埋め込み式のマイクロプロセッサでは、2 進あるいは 10 進の固定小数点数が用いられる。

3　浮動小数点数

符号と有効数字部 (仮数部ともいう) と指数により表した数を**浮動小数点数**という。10 進浮動小数点数 6.02×10^{23} の符号は +(正)、有効数字部は 6.02、指数は 23 である。2 進浮動小数点数 $1.101_{(2)} \times 2^1 (= 3.25_{(10)})$ の有効数字部は $1.101_{(2)}$、指数は 1 である。

計算機では 2 進、10 進、または 16 進の浮動小数点数が用いられる。固定小数点数に比べ演算に時間がかかるが、表せる数の範囲が広くなるという長所があるので、数値計算には通常浮動小数点数が使われる。

3.1　2 進浮動小数点数

2 進浮動小数点数は

$$\pm \left(b_0 + \frac{b_1}{2^1} + \cdots + \frac{b_{p-1}}{2^{p-1}} \right) \times 2^e \qquad (1)$$

と表される。ここで、b_j は 0 または 1、e は整数で $e_{\min} \leqq e \leqq e_{\max}$ である。$b_0 + b_1 2^{-1} + \cdots + b_{p-1} 2^{-p+1}$ は**有効数字部** (仮数部ともいう)、$b_1 2^{-1} + \cdots + b_{p-1} 2^{-p+1}$ は**小数部**、e は**指数**という。$b_0 = 1$ のとき**正規化浮動小数点数** (正規化数と略す) という。ゼロは正規化数ではない。

2 進正規化数の個数は $2^p (e_{\max} - e_{\min} + 1)$ である。最大数は $b_1 = \cdots = b_{p-1} = 1, e = e_{\max}$ のときで、

$$\frac{1 - 2^{-p}}{1 - 2^{-1}} \times 2^{e_{\max}} = 2^{e_{\max}+1} \left(1 - 2^{-p}\right)$$

である。正の最小正規化数は $b_1 = \cdots = b_{p-1} = 0, e = e_{\min}$ のときで $2^{e_{\min}}$ である。

3.2 IEEE754

PC やワークステーションの 2 進浮動小数点数として IEEE754-1985 が標準的に用いられている。昨年 IEEE754-1985 が 23 年ぶりに改訂された。2008 年 6 月に IEEE754-2008 が承認され 8 月に出版された。これまでは、IEEE754-1985 で 2 進浮動小数点数、IEEE854-1987 で 10 進浮動小数点数の規格が定められていたが、IEEE754-2008 ではこれらが統合された。ここでは、IEEE754-1985(IEEE754 と略す) に沿って話していく。

(1) の型の浮動小数点数は符号に 1 ビット (正のとき 0、負のとき 1)、指数部に E ビット、(b_0 は 1 と決まっているので省略し) 小数部に $b_1, \ldots, b_{p-1}$ を格納する。全体で $E+p$ ビットである。

符号	指数部	小数部
1	E	$p-1$

指数部は 0 から 2^E-1 まで 2^E 通りがあり、(1) の指数 e にバイアス $2^{E-1}-1$ を加えた $e+2^{E-1}-1$ を (符号なし)2 進整数で表したものとする。最小指数 $e_{\min}$ は $2-2^{E-1}$、最大指数 $e_{\max}$ は $2^{E-1}-1$ である。したがって、正規化数の指数部は $1 \sim 2^E-2$ となる。指数部の 0 はゼロと非正規化数、2^E-1 は無限大と非数に割り当てる。指数部を非負にすることにより浮動小数点数の大小比較が容易になる。

IEEE754 では 4 種類の形式が規定されているが、よく用いられている単精度と倍精度について次表に示す。

	単精度	倍精度
ビット数	32	64
p	24	53
E	8	11
バイアス	127	1023
$e_{\min}$	-126	-1022
$e_{\max}$	127	1023

たとえば、$3.25_{(10)}$ は単精度では

$$\left(1+\frac{1}{2}+\frac{0}{2^2}+\frac{1}{2^3}+\frac{0}{2^4}+\cdots+\frac{0}{2^{23}}\right)\times 2^1$$

なので、符号 0、指数部 $128_{(10)}=10000000_{(2)}$、小数部 $01010000000000000000000_{(2)}$ である。ビットパタンは 0 10000000 01010000000000000000000 となる。

C 言語は歴史的に $0.5 \leqq$ 有効数字部 < 1 であるので、指数の範囲を IEEE754 より 1 だけ大きくしている [1]。C の単精度では $e_{\min}=-125, e_{\max}=128$ で、バイアスは 126 である。たとえば、$3.25_{(10)}$ は C の単精度では

$$\left(\frac{1}{2}+\frac{1}{2^2}+\frac{0}{2^3}+\frac{1}{2^4}+\frac{0}{2^5}+\cdots+\frac{0}{2^{24}}\right)\times 2^2$$

と表す。ビットパタンは IEEE754 と同一になる。

3.3 IEEE754 での丸め誤差

浮動小数点数は有限個であるので、

1. 実数を浮動小数点数で近似
2. 浮動小数点数同士の四則演算
   (2 つの浮動小数点数を実数と考えたときの和は必ずしも浮動小数点数ではない。)

の際に誤差が入る。これらの端数処理に伴う誤差を**丸め誤差**という。

IEEE754 では、4 種類の丸め方が規定されており、デフォルトは最も近い浮動小数点数に丸める (**最も近い値への丸め**)。最も近い浮動小数点数が 2 つあるときは、小数部の最下位ビットが 0 になるほうを選ぶ (IEEE754-2008 では偶数への丸めと呼んでいる)。

実数 x ($2^{e_{\min}} \leqq |x| \leqq 2^{e_{\max}+1}(1-2^{-p})$) を最も近い値に丸めた浮動小数点数を round(x) と表す。x は

$$x=\pm\left(1+\sum_{j=1}^{\infty}\frac{b_j}{2^j}\right)2^e, \quad e_{\min} \leqq e \leqq e_{\max}$$

と表せるので、round(x) は、

$$\pm(1+b_1 2^{-1}+\cdots+b_{p-1}2^{1-p})2^e \tag{2}$$
$$\pm(1+b_1 2^{-1}+\cdots+b_{p-1}2^{1-p}+2^{1-p})2^e \tag{3}$$
$$\pm 2^{e+1} \tag{4}$$

のいずれかである。(2) は切り捨て、(3) と (4) は切り上げである。とくに、(4) は繰り上がりが生じている。丸め誤差の絶対値の上限は 2^{e-p} である。($(1+2^{-p})2^e$ は 2^e に丸められる。このとき丸め誤差は 2^{e-p} である。)

1 より大きい正規化数の最小のものを $1+\mathrm{eps}$ とする。eps を**マシンエプシロン**という。この定義によると IEEE754 では、$\mathrm{eps}=2^{1-p}$ となる。

丸めの相対誤差 (= (近似値 − 真値)/真値) の絶対値の上限はマシンエプシロンの半分 $\frac{1}{2}\mathrm{eps}$ である。したがって、

$$\mathrm{round}(x) = x(1+\delta), \quad |\delta| \leqq \frac{1}{2}\mathrm{eps}$$

となる。$\frac{1}{2}\mathrm{eps} = 2^{-p}$ を**丸めの単位**という。

3.4 浮動小数点数の例

分かりやすいように、$p = 4, e_{\min} = -1, e_{\max} = 2$ の場合を考える。すべての正の正規化数を 10 進固定小数点数で表したものを次表に示す。有効数字部 (SD) は 2 進固定小数点数、指数は 10 進整数で表す。

SD \ 指数	−1	0	1	2
$1.000_{(2)}$	0.5	1.0	2.0	4.0
$1.001_{(2)}$	0.5625	1.125	2.25	4.5
$1.010_{(2)}$	0.625	1.25	2.5	5.0
$1.011_{(2)}$	0.6875	1.375	2.75	5.5
$1.100_{(2)}$	0.75	1.5	3.0	6.0
$1.101_{(2)}$	0.8125	1.625	3.25	6.5
$1.110_{(2)}$	0.875	1.75	3.5	7.0
$1.111_{(2)}$	0.9375	1.875	3.75	7.5

正規化数は負の数を含め 64 個あり、ゼロに関し対称であるが均等には分布していない。区間 $(-0.5, 0.5)$ には存在せず、$[0.5, 1.0]$ では 0.0625 刻み、$[1.0, 2.0]$ では 0.125 刻み、$[2.0, 4.0]$ では 0.25 刻み、$[4.0, 7.5]$ では 0.5 刻みである。正規化数およびゼロの分布を図 1 に示す。

図 1: 正規化数とゼロの分布

最も近い値への丸めでは、隣り合った正規化数の中点以外は小数部の 2^{-4} の係数 b_4 が 1 のとき切り上げ、0 のときは切り捨てる (零捨一入)。中点のときは b_3 が 0 の方に丸める。丸め誤差の絶対値の上限は $0.0625_{(10)} \times 2^e$ である。マシンエプシロンは $0.125_{(10)} = 1.000_{(2)} \times 2^{-3}$ で、丸めの単位は $0.0625_{(10)}$ である。

4 正規化数の加算

正規化数の加算は次の順序で行う。u, v を正規化数とする。

1. u, v の指数部を比較し、u の指数部が小さいときは u, v を入れ替える。
2. v の指数部を u の指数部に揃え、v の有効数字部を変更する。
3. 加算を行う。
4. 正規化を行う。

アルゴリズムの詳細は [4, pp.204-207] を見よ。指数部を揃える際に小数部 $p-1$ 桁では表現できなくなるので、ほとんどの計算機は 2 桁以上の**保護桁**が用意されている。

例 1 正規化数の加算は $\oplus$ で表すことにする。3.4 節の正規化浮動小数点数 (丸めは最も近い値への丸め) において $(1.001_{(2)} \times 2^1) \oplus (1.001_{(2)} \times 2^0)$ を考える。

$$\begin{array}{rl} & 1.001 \times 2^1 \\ + & 0.1001 \times 2^1 \quad \text{(指数部を揃える)} \\ \hline & 1.1011 \times 2^1 \end{array}$$

であり、$1.1011_{(2)} \times 2^1$ は両隣の正規化数 $1.101_{(2)} \times 2^1$ と $1.110_{(2)} \times 2^1$ の中点なので、小数部の最後のビットが 0 である $1.110_{(2)} \times 2^1$ に丸める。したがって、

$$(1.001_{(2)} \times 2^1) \oplus (1.001_{(2)} \times 2^0) = 1.110_{(2)} \times 2^1$$

である。

保護桁を持たないと

$$\begin{array}{rl} & 1.001 \times 2^1 \\ + & 0.100 \times 2^1 \quad \text{(小数第 4 位は落ちる)} \\ \hline & 1.101 \times 2^1 \end{array}$$

となるので、最も近い値への丸めにならない。

5 桁落ちと情報落ち

5.1 桁落ち

近い数同士の引き算をしたとき、有効桁数が少なくなることを**桁落ち**という。たとえば、

$$1.23456 - 1.23460 = -0.00004$$

は有効桁数 6 桁の数同士の引き算であるが、結果は有効桁数が 1 桁になっている。

桁落ちは式の変形により避けられることがある。

例 2 $|x|$ が十分小さいとき

$$f(x) = \sqrt{1+x} - 1$$

を式の通り計算すると桁落ちが生じるが、分子の有理化

$$f(x) = \frac{x}{\sqrt{1+x}+1}$$

を行うと桁落ちを避けることができる。

次表に $\sqrt{1+x}-1$ と $x/(\sqrt{1+x}+1)$ を示す。有効数字部の数字の下付きの添字は同じ数が添字の数だけ続くことを示す。たとえば、$4.9_4 88 = 4.999988$ である。

x	$\sqrt{1+x}-1$	$x/(\sqrt{1+x}+1)$
1.0×10^{-14}	4.8850×10^{-15}	$4.9_{13}88 \times 10^{-15}$
1.1×10^{-14}	5.5511×10^{-15}	$5.49_{12}85 \times 10^{-15}$
1.2×10^{-14}	5.9952×10^{-15}	$5.9_{13}81 \times 10^{-15}$

$\sqrt{1+x}-1$ の有効桁数は $2 \sim 3$ 桁であるので、$13 \sim 14$ 桁桁落ちしている。$x/(\sqrt{1+x}+1)$ は 15 桁以上正確なので桁落ちはない。

級数展開

$$f(x) \sim \frac{x}{2} - \frac{x^2}{8}$$

によっても同様に桁落ちを避けることができる。

桁落ちは $|x|$ が 0 に近いとき、

$$1 - \cos x, \quad \sin(a+x) - \sin a$$

などでも生じる。それぞれ、

$$1 - \cos x = 2\sin^2 \tfrac{x}{2}$$
$$\sin(a+x) - \sin a = 2 \sin \tfrac{x}{2} \cos(a + \tfrac{x}{2})$$

と変形することにより桁落ちは避けられる。桁落ち防止の変形の多くは、不定形の極限を求めるときに用いる技法と共通している。

$x \to \pm\infty$ のときにも桁落ちが生じることがある。

$$1 + \tanh x = \frac{e^x}{\cosh x}, \quad 1 - \tanh^2 x = \frac{1}{\cosh^2 x}$$

を用いた桁落ち防止の変形を、二重指数関数型数値積分公式 (2008 年 7 月号) で行った。前者は $x \to -\infty$ のとき、後者は $x \to \pm\infty$ のとき桁落ちが生じる。

5.2 情報落ち

絶対値の大きな数に絶対値の小さな数を加えると絶対値の小さな数の情報が反映しない現象を**情報落ち**という。

$1.23456 \times 10^3 + 6.54321 \times 10^{-2}$ を 10 進 6 桁浮動小数点数として計算する。指数を 10^3 に揃え保護桁を 2 桁取り有効数字部を計算する。

	1.23456	10^3
+	0.0000654	10^3
	1.2346254	10^3

有効数字部の小数第 6 位を四捨五入して

$$1.23456 \times 10^3 + 6.54321 \times 10^{-2} = 1.23463 \times 10^3$$

が得られる。6.54321×10^{-2} の有効数字部 0.04321 が結果に反映していない。

例 3 級数 $\sum_{i=1}^{\infty} i^{-2} = \pi^2/6$ の部分和を

$$s_\nu = \sum_{i=1}^{\nu} \frac{1}{i^2}$$

とおき、s_{8000} を 2 通りの方で計算する。計算は単精度 (10 進 7.2 桁) である。

$$((((\frac{1}{1^2} + \frac{1}{2^2}) + \frac{1}{3^2}) \cdots) + \frac{1}{8000^2})$$

により計算すると 1.6447253 となる。$\nu \geqq 4096$ のとき $s_\nu = 1.6447253$ である。これは、

$$\frac{1}{4096^2} = 2^{-24} = \frac{1}{2}\mathrm{eps}, \quad 1 < s_\nu < 2$$

であるので、$\nu \geqq 4097$ のとき $1/\nu^2$ は結果に反映していない。一方、加える順序を逆にして

$$((((\frac{1}{8000^2} + \frac{1}{7999^2}) + \frac{1}{7998^2}) \cdots) + \frac{1}{1^2})$$

により計算すると 1.6448090 と情報落ちは回避できる。

級数の和を高精度に計算する方法に**カハンの加算アルゴリズム**がある。浮動小数点数 $x_1, \cdots, x_\nu$ の和 $\sum_{i=1}^{\nu} x_i$ を次のアルゴリズムで計算する。[2, pp.83-88]

$s = 0$
$e = 0$
for $i = 1 : \nu$
  $tmp = s$
  $y = x_i + e$
  $s = tmp + y$
  $tmp = tmp - s$
  $e = tmp + y$
end

定理 1 (クヌース [4]) カハンの加算アルゴリズムで計算した和 s は

$$s = \sum_{i=1}^{\nu}(1+\delta_i)x_i, \quad |\delta_i| \leqq 2u + O(\nu u^2)$$

を満たす。ここで、$u = 2^{-p}$ は丸めの単位である。

証明 [4, p.230,p.537][2, p.85] を見よ。 □

例 4 カハンの加算アルゴリズムにより

$$s_{8000} = ((((\frac{1}{1^2} + \frac{1}{2^2}) + \frac{1}{3^2})\cdots) + \frac{1}{8000^2})$$

を単精度で計算すると $s_{8000} = 1.6448091$ となり、情報落ちは回避できる。

カハンの加算アルゴリズムにおいて、$s = \sum_{j=1}^{i-1} x_j$ に $x = x_i$ を加えるところの有効数字部の変化を図解する。有効数字部を p ビットとし、s と x の指数の差が q であるとする。図 2 では指数部を揃え、有効数字部に $2p$ ビット使っている。s の上位の q ビットを $s^{(1)}$ とし、下位の $p-q$ ビットを $s^{(2)}$ とする。x の上位の $p-q$ ビットを $x^{(1)}$、下位の q ビットを $x^{(2)}$ とする。e の上位の $p-q$ ビットを $e^{(1)}$、下位の q ビットを $e^{(2)}$ とする。桁数を揃えるため、必要なときは上位や下位ビットを 0 で埋める。

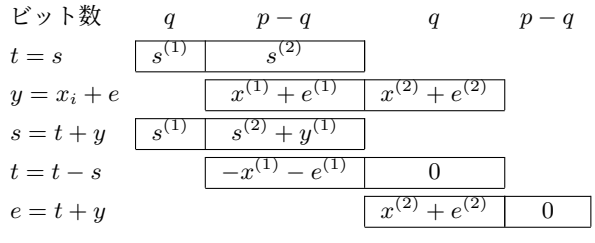

図 2: カハンの加算アルゴリズム

$s = t + y$ の際に情報落ちとなる $x^{(2)} + e^{(2)}$ は e に格納される。$y = x_i + e$ における情報落ちのため、e は正確な補正ではない。

筆者の同僚にベクトルの内積の高精度計算 [5] などの研究をしている荻田武史さんがいる。彼に尋ねた所、カハンのアルゴリズムにより倍精度 (64 ビット) で計算する際、浮動小数点レジスタが 64 ビットの場合はうまく働くが、浮動小数点演算の誤差を利用しているため浮動小数点レジスタが 80 ビットの場合は問題が生じることがあるという。

ためしに、倍精度で

$$((((\frac{1}{1^4} + \frac{1}{2^4}) + \frac{1}{3^4})\cdots) + \frac{1}{16000^4})$$

をカハンのアルゴリズムにより計算してみた。浮動小数点レジスタが 80 ビットである Vine Linux 3.3 上 gcc 3.3.2 と Cygwin 上 gcc 3.4.4 を用いたが、いずれもカハンのアルゴリズムは正常に働いた。

6 関孝和の円周率の計算の誤差

連載の第 5 回 (2008 年 9 月号) で関孝和の円周率の計算を取り上げた。直径 1 の円に内接する正 2^ν 角形の周の長さを s_ν とする。「s_{15}, s_{16}, s_{17} の小数第 17 位以下は桁落ち (近い数同士の引き算により有効桁数が少なくなる現象) などのため正しくない。」と書いたが、厳密にいうと桁落ちとも情報落ちとも異なり、**固定小数点数計算で小さい数を扱ったことによる有効桁数の減少**が原因である。

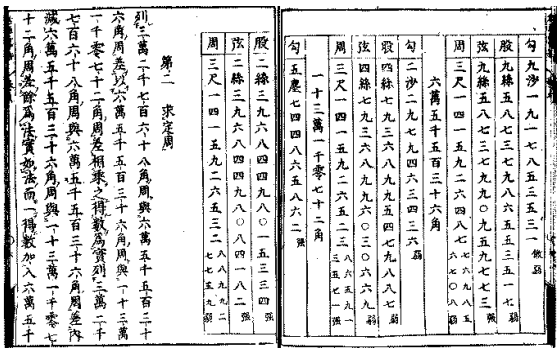

図 3:『括要算法』(東北大学・岡本刊 089)

関の没後に出版された『括要算法』[7] に計算結果 (図 3) が記載されている。左側のページの 3 行目までは周の計算、「第二 求定周」ではエイトケン Δ^2 法

と呼ばれている加速法により円周の長さを定めている。一絲 (し) は一尺 (約 30.3cm) の 10^5 分の一、一塵 (じん) は一尺の 10^{10} 分の一である。直角三角形の斜辺を弦、直角を挟む短い辺を勾 (こう)、長い辺を股 (こ) という。

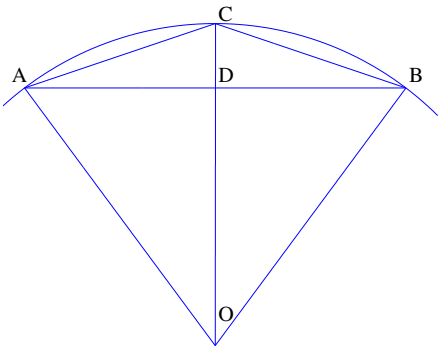

図 4: 正 2^ν 角形の勾股弦

図 4 において、AB は正 $2^{\nu-1}$ 角形の弦、CD、AD と AC はそれぞれ正 2^ν 角形の勾、股、弦である。

規則性を見て取れるように、$\nu = 17$(一十三萬一千零七十二角) の場合に一尺を 1 とする固定小数点数で表した値を次表に示す。

	固定小数点数表示
勾	0.00000 00005 74486 5862 強
股	0.00002 39684 49801 5334 強
弦	0.00002 39684 49808 4182 強
周	3.14159 26532 88992 7759 弱

関は小数第 20 位を丸めて第 19 位まで (整数部 1 桁、小数部 19 桁の固定小数点数により) 表していることが分かる。

小数第 d 位まで計算したとすると、勾の有効桁数は $d-9$ である。関は

$$\text{弦} = \sqrt{\text{勾}^2 + \text{股}^2}, \quad \text{周} = 2^\nu \text{弦}$$

によって周を求めたので、周の有効桁数も $d-9$ 程度となる。周の有効桁数は 17 であるので、勾の有効桁数も 17 桁前後と考えられる。つまり関の誤差は、有限桁の固定小数点数計算において計算の途中で絶対値の小さい数が現れたことによる有効桁数の減少が原因、ということになる。

有効桁数の減少を手がかりにして、関が図 3 の値を小数第 26(= 17 + 9) 位までの計算により求めたことが示されている [6]。

参考文献

[1] 日本工業標準調査会、プログラム言語 C JIS X 3010:2003
("ISO/IEC 9899:1999 - Programming Language C"の日本語訳。)
http://www.jisc.go.jp/

[2] N.J. Higham, Accuracy and Stability of Numerical Algorithms, 2nd. ed., SIAM, 2002.

[3] IEEE, Draft Standard for Floating-Point Arithmetic P754, 2006.
http://isavhdlimp.sourceforge.jp/pdf/IEEE754r-2006-10-04.pdf

[4] D.K. クヌース著、有沢・和田監訳、The Art of Computer Programming, Vol.2, Seminumerical Algorithms, 3rd. ed. 日本語版、ASCII, 2004

[5] 荻田武史、品質を落とさない数値計算法、数学セミナー、2008 年 11 月号、15-19

[6] 長田直樹、関孝和の円周率の計算、京都大学数理解析研究所講究録、近刊
http://www.cis.twcu.ac.jp/~osada/rims2008.pdf

[7] 関孝和、括要算法、東北大学和算ポータル、岡本刊、請求番号 089
http://www2.library.tohoku.ac.jp/wasan/

(おさだ なおき/東京女子大学)