半正定値計画法に対する高精度・安定計算の技術

中田　真秀

半正定値計画問題（SDP）は，理論もプログラムも整備されているが，実際には解の精度が出なかったり，計算が不安定になることがある．極端な場合は計算によって得られたそれらしき解が全く間違っていることさえある．この状況を改善するために高精度SDPソルバーを開発してきた．これらはSDPを任意精度（SDPA-GMP），疑似8倍精度（SDPA-QD），疑似4倍精度（SDPA-DD）で解く．また同時期に多倍長精度版の線形代数パッケージである，MPACK（MBLAS/MLAPACK）も開発してきた．これらについて解説する．

キーワード：半正定値計画法，多倍長精度計算，BLAS，LAPACK，MPACK

1．半正定値計画は解に近づくにつれ高精度に解くのが難しくなる

ここでは，半正定値計画問題（SDP）について，正確に解くにはどこで，なぜ高精度計算が必要になるかについて説明し，実装および数値実験を示す．主双対内点法は非常に正確にSDPを解く方法として知られている．しかし実際には解に近づくにつれ精密に解くのは難しくなる．その理由は，大雑把には，多くの場合最適解付近では解の行列の条件数が大きくなり，最適解では発散してしまうからである．行列の条件数が発散するとなぜ困るかであるが，反復の際に現れる線形方程式の解の誤差が大きくなってしまうのである．この誤差の蓄積で，正確に解きにくくなる．よく使われる倍精度では16桁の精度があるが，これを使って最適解を8桁以上の精度を出すのは難しい．そこで高精度計算が必要になる．

例を一つ挙げよう．グラフ分割問題に対するあるSDP緩和問題を解いたときの最適値が高精度版のSDPA-GMPでは，−1.1957989287 e-01と計算される．しかし倍精度版であるSDPAでは最適解を−1.1972 e-01から−1.2187 e-01の間と誤った計算をしてしまうのだ[5]．

さて，ここでは，本特集の中の中田和秀氏による「半正定値計画の問題記述&解決能力」の定義を用いる．SDPの主，双対問題の定式化については2節「SDPとは」，主双対内点法については4.1節「主双対内点法」を参考いただきたい．その4節にもある通り，一般的なSDPをある程度の精度で解く場合には，2階の微分情報を利用した解法の方が安定して最適解を求めることができる．そして多くの主双対内点法の実装ではそれを用いている．その箇所はStep 2に挙げられている，Newton法で探索方向を計算するところである．

しかし，それにもかかわらず高精度の解を得るのは難しいことについて説明しよう．最適解の条件に相補性定理というのがある．もしSDPが（主），（双対）について内点許容解をもつ場合［つまり$(\boldsymbol{x}^*, \boldsymbol{X}^*, \boldsymbol{Y}^*)$が$(\boldsymbol{X}^* \succ \boldsymbol{0}, \boldsymbol{Y}^* \succ \boldsymbol{0})$を満たし，すべての線型拘束条件を満たす］$(\boldsymbol{X}^*, \boldsymbol{Y}^*)$が最適解である必要十分条件は，

$$\boldsymbol{X}^* \boldsymbol{Y}^* = \boldsymbol{0} \tag{1}$$

となる．したがって$\boldsymbol{X}^*$，$\boldsymbol{Y}^*$が最適解のとき，線形代数の定理から

$$\mathrm{rank}\boldsymbol{X}^* + \mathrm{rank}\boldsymbol{Y}^* \leq n \tag{2}$$

となる．これから$\boldsymbol{X}^*$，$\boldsymbol{Y}^*$の少なくとも一方はrankがn未満になる必要がある．つまり，$\boldsymbol{X}^*$，$\boldsymbol{Y}^*$の少なくとも一方には逆行列は存在しない．しかし，反復点$\boldsymbol{X}$，$\boldsymbol{Y}$がそれぞれ$\boldsymbol{X}^*$，$\boldsymbol{Y}^*$に近付く計算の過程には$\boldsymbol{X}$，$\boldsymbol{Y}$の逆行列が必要であり，$\boldsymbol{X}^*$，$\boldsymbol{Y}^*$に近付くほど$\boldsymbol{X}$，$\boldsymbol{Y}$の逆行列を有限桁数で精度よく求めるのは難しくなる．$\boldsymbol{X}$，$\boldsymbol{Y}$の逆行列が正確に計算できない結果，それらの値を用いるSchur補完行列$\boldsymbol{B}$にも誤差が生じ，正確な計算からずれてしまうのである．

残念ながら，この問題は本質的である．もちろん，問題ごとにSDPの定式化をうまく行えば精度を高められる場合もあるが，一般の処方で避けることはできない．

なかた　まほ
理化学研究所 情報基盤センター
〒351-0198 和光市広沢2-1

我々は高精度計算で，強引にではあるが，この問題の一部を解決する．通常コンピュータでの数値演算は，浮動小数点というフォーマットを用いその上での加減乗除を定義して行われる．ANSI/IEEE Standard 754-2008という規格[8]があり（実際はまだ旧版1985年版であろうが）ほとんどのコンピュータはこれをハードウェアでサポートし，また演算速度も高速である．そのフォーマットの一つであるbinary 64，いわゆる倍精度というものは約16桁の有効桁をもち，頻繁に用いられている．それを任意に大きくしたり，64桁にしたり，32桁にしたりして高精度な計算を行う．そのようにSDPA-GMP（任意），SDPA-QD（64桁），SDPA-DD（32桁）を開発してきた．実際にこれらのSDPソルバーをつかえば非常に高精度に解けるようになった．SDPA-GMPの最初のバージョンは2008/4/10に，そしてSDPA-QD，-DDは2009/3/22に公開された．すでにこれらを使った研究がいくつかある．また，SDPAでは線形代数演算を多数行うが，これにはBLAS, LAPACKというライブラリを使っている[10][1]．SDPA-GMP，-QD，-DDに対応するライブラリは存在しなかったため多倍長精度版のBLAS，LAPACKであるMPACK（MBLAS/MLAPACK）を別途開発した[11]．高精度演算ができるようになったが，演算速度は犠牲になった．従来のものより数十倍から数千倍遅くなってしまった．以下SDPA-GMP，-QD，-DDの解説を行う．構成は，まず浮動小数点演算と多倍長精度演算ライブラリについて，次にMPACK（MBLAS/MLAPACKについて），さらにSDPA-GMP，-QD，-DDの実装について，数値計算の評価，最後に結論という順序になっている．

2. 浮動小数点演算と多倍長精度演算ライブラリについて

コンピューター上で実数を扱うのは難しいため，計算を行うための次のような「浮動小数点フォーマット」というのがよく使われる．

$$(-1)^s \times 2^e \times m, \tag{3}$$

ここでsは符号（0または1），eは指数部，そしてmは次のような文字列で合わされるような数で，

$$m = d_0.d_1d_2\cdots d_{p-1}, \tag{4}$$

d_iは0または1（したがって$0 \leq m < 2$となる），そしてpは有効桁である[9]．この上に演算は定義されるが，演算には誤差が一般には含まれる（丸め誤差）．これが数値演算の誤差の原因であり，数値的な不安定性を引き起こしたりする．以下ANSI/IEEE Standard 754-2008，GMPライブラリ，QDライブラリについて説明する．

2.1 ANSI/IEEE Standard 754-2008；浮動小数点演算の標準規格

浮動小数点フォーマットとその上の演算の定義の仕方は様々であるので，システムが違っていればそれごとに異なっていた．1985年にANSI/IEEE Standard 754-1985が発効し，この混沌は解決された．現在ほぼすべてのコンピュータがこれに準拠しており，さらに最近2008年版も発効された[8]．さてこの最新のANSI/IEEE Standard 754-2008のbinary 64という浮動小数点フォーマットがある（以下IEEE 754倍精度とする）．IEEE 754倍精度のeは(3)式において$-1022 \leq e+1023$であり(4)式におけるpは53なので，16桁の精度（$15.954 \simeq 53\log_{10}2$）を持ち，最もよく使われている．その理由は(i)非常に高速であるから；例えば理論性能値はNVIDIAのTesla C 1060は78 GFlops，Core i7 975は55.4 GFlopsとなっている（GFlops＝1秒に10億回浮動小数点演算実行），(ii)大抵の用途では16桁の精度で十分である，からだ．

2.2 任意精度：The GNU Multiple Precision Arithmetic Library

GNU Multiple Precision Arithmetic Library（GMP）は任意精度の計算ライブラリであり，符号つき整数，有理数，浮動小数点が扱える[6]．GMPでは式(3)の指数部eが$2^{-32} \leq e \leq 2^{32}$または$2^{-64} \leq e \leq 2^{64}$となっておりとても大きい．さらに式(4)の仮数部$p$は32または64の倍数で任意に大きさを指定できる．SDPA-GMPでのpのデフォルトは250(256)であり，約77桁の有効桁をもつ（$77.06 \simeq 256\log_{10}2$）．したがって，驚異的に正確な演算が可能となる．それに加えC++の言語バインディングを用いるのが高精度SDPAの開発に重要かつ本質である．その理由はプログラム作成が簡略化，共通化されるからである．複雑な行列，ベクトル演算をSDPAでは多用しており，高級な書き方は必須である．これでC/C++の“double”とほとんど変わらないプログラムが可能となる（図1）．

2.3 疑似4,8倍精度：Quad-Doubleライブラリ

Quad-Double（QD）ライブラリは，Y. Hidaらの開発した[7]多倍長精度計算ライブラリの一つである．Knuth[9]，Dekker[3]による巧妙な浮動小数点演算

```
#include <iostream>
#include <gmpxx.h>

int main()
{
  mpf_class a, b;
  a = 1.0;
  b = 2.0;
  std::cout << "a + b = " << a + b << "\n";
  std::cout << "sqrt (a) = " << sqrt (a) << "\n";
}
```

図1　GMPとC++バインディングを用いたサンプル．C/C++の“double”と同じように簡単に組める．

における和，積の誤差の厳密な分離定理を用いたものであり，浮動小数点演算しか用いないため，GMPに比べて簡単，高速になる場合も多い．基本となるアイディアは幾つかのIEEE 754倍精度をそのまま，未評価のまま配列として持っておくということである．二つの型QD型とDD型があり，IEEE 754倍精度をそれぞれ4個，2個持っているため，QD型は約64桁，DD型は約32桁の精度を持つ．例えばQD型の数 a は，(a_1, a_2, a_3, a_4) として表現される：

$$a = a_1 + a_2 + a_3 + a_4. \tag{5}$$

DD型 b はIEEE 754倍精度2個で (b_1, b_2)

$$b = b_1 + b_2. \tag{6}$$

と表現される．次に演算の仕組みを簡単に紹介する．$\oplus$, $\ominus$, $\otimes$をIEEE 754倍精度の間の和，差，積としよう．このとき，一般には計算に誤差が混入するのだが，驚くべきことにその誤差は実は厳密に評価できるのである．二つのIEEE 754倍精度の数 x, y をとる．次のようにすると $s=(x\oplus y)$ および $e=x+y-(x\oplus y)$ を**厳密**に評価できる[9][12]：

Two-Sum (x, y)

1. $s \leftarrow x \oplus y$
2. $v \leftarrow s \ominus x$
3. $e \leftarrow (x \ominus (s \ominus v)) \oplus (y \ominus v)$
4. **return** (s, e)

積についても同様のことができる．まずIEEE 754倍精度 x を重なりあわない二つの数 x_{hi}, x_{lo} に分ける[3][12]：

Split (x)

1. $t \leftarrow (2^{27}+1) \otimes x$
2. $x_{hi} \leftarrow t \ominus (t \ominus x)$
3. $x_{lo} \leftarrow x \ominus x_{hi}$
4. **return** (x_{hi}, x_{lo})

次に二つのIEEE 754の厳密な積を行う[12]：

Two-prod (x, y)

1. $p \leftarrow x \otimes y$
2. $(x_{hi}, x_{lo}) \leftarrow$ Split (x)
3. $(y_{hi}, y_{lo}) \leftarrow$ Split (y)
4. $e \leftarrow ((x_{hi} \otimes y_{hi} \ominus p) \oplus x_{hi} \otimes y_{lo} \oplus x_{lo} \otimes y_{hi}) \oplus x_{lo} \otimes y_{lo}$
5. **return** (p, e)

そして，$x \times y = x \otimes y + e = p + e$ とする．Two-Prodも**厳密**な変換である．

さて，DD型の二つの数 b, c に和 $d=b+c$ を定義しよう．$d=(d_1, d_2)$，$b=(b_1, b_2)$，$c=(c_1, c_2)$ となる．さらに t_1, t_2 をIEEE 754倍精度とすると[7]：

Double-double-addition (b, c)

1. $(d_1, d_2) \leftarrow$ Two-sum (b_1, c_1)
2. $(t_1, t_2) \leftarrow$ Two-sum (b_2, c_2)
3. $d_2 \leftarrow d_2 \oplus t_1$
4. $(d_1, d_2) \leftarrow$ Two-sum (d_1, d_2)
5. $d_2 \leftarrow d_2 \oplus t_2$
6. $(d_1, d_2) \leftarrow$ Two-sum (d_1, d_2).

となる．差，積，商に関するアルゴリズムも同じようにして行う[7]．GMPのようにQDにもC++バインディングがある．注意すべき点はDD型もQD型もIEEE 754倍精度の拡張ではないことである．有効桁は増えているが，指数部は短くなり，IEEE 754倍精度で表現される最小の数をDD，QD型では表すことができない．したがってアンダーフローが起こりやすい．

3. MPACK（MBLAS/MLAPACK）：多倍長精度版BLAS，LAPACK

BLASはFORTRAN 77で書かれ高品質な基本的なベクトル，行列演算の「ビルディングブロック」ルーチンを提供する[10]．レベルが3つあり，ベクトル-ベクトル（level 1 BLAS），ベクトル-行列（level 2 BLAS），行列-行列（level 3 BLAS）演算を行う．移植性がよく，効率も高いので，最も広く使われている．LAPACKはFotran 90で書かれており，連立一次方程式，最小2乗法，固有値問題，特異値分解などを行う[1]．MPACK（MBLAS, MLAPACK）は多倍長精度版のBLASとLAPACK[11]であり，中田によって開発されてきた．開発のポリシーは：

- BLAS, LAPACKに近いAPIを提供する．
- C++で書く．
- 移植性を高くする：OSや多倍長精度演算ライブラリに依存しないようにする．

であり，2010/1/13にリリースされた0.6.4の状況は

- LAPACK 3.1.1をベースとする．

- MBLAS 76 の全ルーチンが利用可能.
- MLAPACK の 90 のルーチンが利用可能.
- 逆行列，LU 分解，コレスキー分解，対称，エルミート行列の固有値問題などに対応.
- GMP/QD が多倍長精度演算ライブラリとして利用可能.

となっている．もちろん BLAS, LAPACK 同様，MPACK をベースとした最適化 MBLAS, MLAPACK があってもよく，我々の開発計画にも入っている.

4. 高精度 SDPA の実装

現在，SDPA-GMP, -QD, -DD は SDPA 7.1.2 をオリジナルとしている．SDPA からの変換は非常にシンプルであった．大体の場合ソースコードの “double” を対応する多倍長精度演算クラスである “mpf_class” (SDPA-GMP), “qd_real” (SDPA-QD), “dd_real” (SDPA-DD) に機械的に変換するだけで良かった．一番大きな違いは，SDPA では BLAS や LAPACK を呼び出すところの代わりに多倍長精度版は MBLAS, MLAPACK を呼び出すところである．API はお互い似ているため，両者の違いは最小限となっている．実際開発のコストは MBLAS, MLAPACK の方がはるかに高い．図 2 および 3 を比較するとわかりやすいだろう.

```
bool Lal::getInnerProduct(double& ret, Vector& aVec,
 Vector& bVec)
{
  int N = aVec.nDim;
  if (N != bVec.nDim) {
    rError("getInnerProduct:: different memory
 size");
  }
  ret = ddot_f77(&N,aVec.ele,&IONE,bVec.ele,&IONE);
  return _SUCCESS;
}
```

図 2 SDPA 7.1.2 からソースコード抜粋．(sdpa_linear.cpp) これを元にしている.

```
bool Lal::getInnerProduct(mpf_class& ret, Vector&
aVec, Vector& bVec)
{
  int N = aVec.nDim;
  if (N != bVec.nDim) {
    rError("getInnerProduct:: different memory
 size");
  }
  ret = Rdot(N, aVec.ele, 1, bVec.ele, 1);

  return _SUCCESS;
}
```

図 3 SDPA-GMP 7.1.2 からソースコード (sdpa_linear. cpp) の抜粋．多倍長精度化によるコードの変更点は少ない.

5. 高精度 SDPA の利用上の注意点

まず，計算時間が数百から数千倍遅くなっている，ということである．これは多倍長精度演算がハードウェアサポートされていないことによる．また，入力ファイルからはより長い浮動小数点フォーマットを読み込むことが可能になった．この点には注意せねばならないだろう．もし 1/3 を 0.3333333333333333 と表現していたら，0.3333…3333 と適切に長くしたほうが良いだろう．SDPA では 16 桁以降は無視されるが，SDPA-GMP, -QD, -DD などはそれ以上の桁を読める．SDPA-GMP の場合パラメータファイルも新しいパラメータ “precision” が導入されている (デフォルトで 250；約 77 桁)．これは精度をビット数で入力し，計算で使う精度を指定する．SDPA, SDPA-QD, -DD では精度は固定なのでそのようなパラメータは必要ではない．指定しても単に無視されてしまう．したがってインプットファイルは共通化できる.

6. SDPLIB の問題の結果

まず SDPLIB[2] から well-posed[4] な問題，つまり主双対内点法の枠組みが成り立つはずの系である maxG 11，および truss 8 の応用を SDPA-GMP, -QD, -DD を使って行った．まず高精度 SDPA の実装の正しさ，内点法の正しさなどを数値的に確認するためである．パラメータ epsilonStar, epsilonDash, “precision” (SDPA-GMP のみ) はそれぞれ 1 e-30, 1 e-30, 250 とした．SDPA ではそれぞれ 1 e-9, 1 e-9

表 1 SDPLIB から well-posed な問題を SDPA-GMP (77 桁), -QD (64 桁), -DD (32 桁), SDPA (16 桁) を用いて解いたときの最適値，双対ギャップ，主双対実行可能誤差，反復回数．括弧内は精度.

問題	SDPA-GMP	SDPA-QD	SDPA-DD	SDPA
maxG11	最適値:6.2916478300199902e+02			
双対ギャップ	3.51e-31	3.51e-31	3.51e-21	3.47e-08
主実行可能誤差	1.65e-76	8.55e-65	9.24e-32	6.66e-16
双対実行可能誤差	6.07e-74	3.95e-55	4.12e-26	6.44e-14
反復回数	39	39	29	16
時間 (秒)	22236.9	19756.2	1926.4	15.5
truss8	最適値:-1.3311458915226341e+02			
双対ギャップ	2.27e-31	2.27e-31	2.27e-21	1.35e-08
主実行可能誤差	1.55e-76	4.86e-63	6.31e-30	9.12e-09
双対実行可能誤差	2.41e-75	4.53e-59	1.40e-26	2.54e-10
反復回数	44	44	34	20
時間 (秒)	434.1	356.3	37.1	3.2

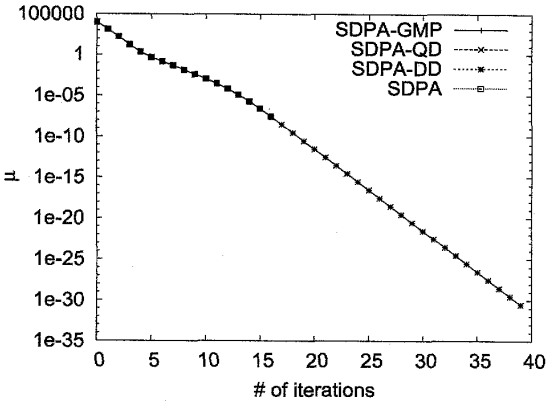

図4 SDPLIBからwell-posedなmaxG 11のμと反復回数の図；SDPA-GMP，-QD，-DD，SDPAの収束の振る舞いはほとんど同じであるが，精度の関係でSDPA，-DD，-QD，-GMPの順に終了する．

とした．表1に数値実験の結果を示した．最適値，双対ギャップ，主および双対実行可能誤差，反復回数，時間をSDPA-GMP，-QD，-DD，SDPAで示した．maxG 11, truss 8の最適値はSDPA-GMPでそれぞれ6.2916478300199902 e+02，−1.3311458915226341 e+02であった．双対ギャップはSDPA-GMP，-QDで1 e-30以下，-DDで1 e-20だったが，SDPAでは1 e-8であった．SDPA-GMPでは主双対両最適値は16桁以上一致した．主双対実行可能誤差も非常に小さかった．パラメータμと反復回数を図4に示した．μは反復中$\boldsymbol{X}(\mu)\boldsymbol{Y}(\mu)=\mu\boldsymbol{I}$となるようになっており，$\ln\mu$は反復回数にほぼ比例するように小さくなってゆく．この場合μはソルバーによらず理論から予想されるのとほぼ同じように小さくなっていった．maxG 11の反復回数はSDPA-GMP，-QD，-DD，SDPAでそれぞれ39,39,29,16回であった．ここでは示さないが他のwell-posedな問題でも似たような振る舞いであった．

次にill-posedな問題[4]つまり内点が（双対）側にない問題であるgpp 124-1，gpp 250-1，gpp 250-4，gpp 500-1，qap 5，qap 8，qap 10，hinf 3，hinf 6を解いた．表2に結果を示す．この場合主双対内点法は収束するという保証がないにもかかわらず非常に正確な最適解を出すことができた．例えばSDPA-GMPでのgpp 124-1の最適値を示すと−7.3430762652465377であった，主双対両方16桁一致した．双対ギャップは非常に小さく，例えばqap 5で9.49 e-30などであった．図5には収束のプロファイルを示した．これは通常の振る舞いのように見えるが，SDPA-GMPと-QDの結果は11桁までしか一致していない；

表2 SDPLIBからill-posedな問題をSDPA-GMP（77桁），-QD（64桁），-DD（32桁），SDPA（16桁）を用いて解いたときの最適値，双対ギャップ，主双対実行可能誤差，反復回数．括弧内は精度．

問題	SDPA-GMP	SDPA-QD	SDPA-DD	SDPA
gpp124-1	最適値:-7.3430762652465377			
双対ギャップ	7.06e-78	1.74e-22	2.74e-10	7.33e-08
主実行可能誤差	2.85e-76	2.08e-46	3.97e-23	3.64e-12
双対実行可能誤差	2.62e-34	2.55e-21	2.82e-10	1.16e-08
反復回数	57	99	27	19
時間（秒）	296.0	422.3	14.9	0.4
gpp250-1	最適値:-1.5444916882934067e+01			
双対ギャップ	1.68e-78	1.33e-23	4.62e-11	5.87e-08
主実行可能誤差	2.25e-76	5.08e-47	3.31e-24	6.82e-13
双対実行可能誤差	6.07e-35	6.36e-22	3.18e-11	3.39e-08
反復回数	62	101	28	23
時間（秒）	2589.0	3428.5	120.7	1.2
gpp250-4	最適値:-7.4732831101269269e+02			
双対ギャップ	5.13e-20	5.09e-20	1.22e-11	5.52e-09
主実行可能誤差	5.35e-76	3.68e-51	3.93e-24	5.46e-12
双対実行可能誤差	7.57e-31	7.53e-31	2.09e-11	3.88e-11
反復回数	76	76	28	21
時間（秒）	3217.1	2552.0	123.3	1.6
gpp500-1	最適値:-2.5320543879075787e+01			
双対ギャップ	5.95e-78	2.31e-25	1.22e-11	1.43e-08
主実行可能誤差	3.20e-76	6.57e-47	2.48e-24	6.82e-13
双対実行可能誤差	1.63e-31	2.90e-21	9.59e-12	3.07e-10
反復回数	73	172	30	22
時間（秒）	22937.4	45174.8	1031.3	6.6
qap5	最適値:-4.3600000000000000e+02			
双対ギャップ	9.49e-30	6.78e-20	5.04e-09	6.94e-05
主実行可能誤差	2.53e-75	8.56e-52	4.65e-25	1.46e-11
双対実行可能誤差	7.80e-42	9.63e-31	1.16e-13	7.88e-07
反復回数	50	36	19	15
時間（秒）	14.4	8.5	0.6	0.2
qap8	最適値:-7.5695525603861951e+02			
双対ギャップ	1.06e-17	9.28e-17	1.42e-09	2.91e-04
主実行可能誤差	4.30e-75	1.55e-49	8.27e-25	4.37e-11
双対実行可能誤差	9.09e-31	7.42e-29	1.16e-12	9.05e-07
反復回数	123	111	27	14
時間（秒）	1148.6	851.1	25.8	0.6
qap10	最適値:-1.0926074684462389e+03			
双対ギャップ	6.91e-18	6.93e-17	6.74e-08	6.89e-04
主実行可能誤差	5.01e-75	9.09e-50	1.03e-24	3.49e-10
双対実行可能誤差	6.16e-31	7.32e-29	1.57e-12	6.71e-06
反復回数	96	87	20	15
時間（秒）	5788.8	4331.3	125.1	2.0
hinf3	最適値:5.6940778009669388e+01			
双対ギャップ	6.63e-31	6.63e-31	2.90e-08	2.93e-03
主実行可能誤差	1.55e-75	1.27e-57	2.83e-27	1.82e-12
双対実行可能誤差	1.90e-41	1.90e-41	9.08e-10	2.19e-05
反復回数	141	141	45	26
時間（秒）	0.7	0.7	0.0	0.0
hinf6	最適値:4.4892774532353916e+02			
双対ギャップ	2.38e-31	2.38e-31	1.25e-09	4.36e-05
主実行可能誤差	4.89e-75	3.00e-56	7.75e-24	1.46e-11
双対実行可能誤差	3.51e-60	1.15e-41	1.11e-12	1.53e-05
反復回数	76	76	53	20
時間（秒）	0.4	0.4	0.0	0.0

5.6940778009669388 e+01，5.6940778009084397 e+01．双対ギャップ（6.63 e-31）や他の誤差は非常に小さく，正確さの判断が難しい．

図6にgpp 250-1の結果を示す．これは見るからに

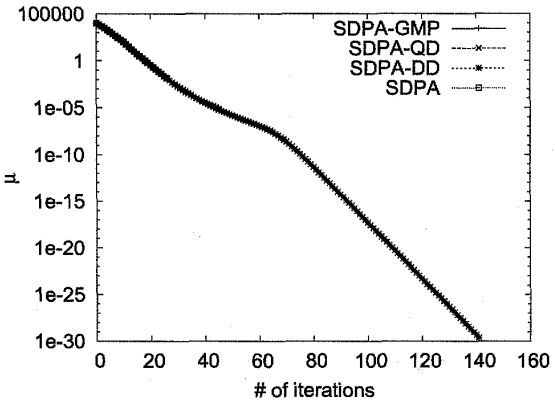

図5 SDPLIBからill-posedなhinf 3のμと反復回数の図；振る舞いは良さそうに見えるが最適値は少し正確さに欠ける．

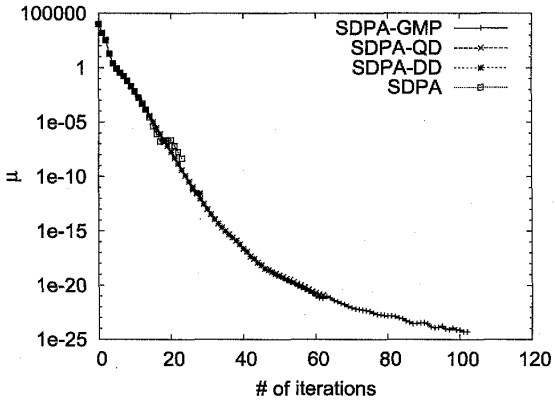

図6 SDPLIBからill-posedなgpp 250-1のμと反復回数の図；振る舞いは不安定．

不安定である，解付近になればなるほど収束が緩慢になってくる．ただし求まった値は悪くはなかった．またhinf 2，hinf 3，hinf 6以外のhinf系，hinf 1，hinf 4，hinf 5などでは最適値が求まらなかった．パラメータepsilonStar，epsilonDashに依存した，見せかけの最適値が求まった．この際も主双対実行可能誤差は十分に小さかった．

いずれの場合もSDPA-GMP，SDPA-QDで解く時間は長い．精度が高くなるにつれ反復回数も増えるため単純な比較はできないが，例えばgpp 250-1では2,000倍遅かった．SDPA-DDは100倍程度遅かった．正確さと時間の兼ね合いで使い分けるのが良いだろう．計算はCPU：Opteron 250 2.4 GHz，メモリ：16 G，OSはFreeBSD 7/amd 64という構成で行った．

7. おわりに

ここではSDPA-GMP，-QD，-DDという多倍長精度演算ライブラリを用いた高精度SDPAソルバーおよび多倍長精度版のBLAS，LAPACKであるMPACKを紹介した．主双対問題に内点がある場合は，理論どおり収束しかつ非常に正確に解けた．そうでない場合も大抵の場合非常に正確に解けた．が，収束の振る舞いは変則的なものもあった．収束しづらいのもあった．解くのにかかる時間はSDPA-GMP，-QDでは数千倍遅く，SDPA-DDでは百倍程度遅かった．用途によって使い分けると良いだろう．今後については，さらなる応用，より高速な実装，理論的な解析などに取り組みたい．

参考文献

[1] E. Anderson, Z. Bai, C. Bishof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. Mckenney and D. Sorensen, *LAPACK Users' Guide, third edition*, SIAM, Philadelphia, PA, USA, 1999.

[2] B. Borchers, *Optim. Methods and Softw.*, Vol. 11 (1), 1999, pp. 683-690.

[3] T. J. Dekker, *Numerische Mathematik*, Vol. 18, 1971, pp. 224-242.

[4] R. M. Freund, F. Ordonez and K. -C. Toh, *Math. Prog. B*, Vol. 109, 2007, pp. 445-475.

[5] 藤澤克樹, 梅谷俊治, 応用に役立つ50の最適化問題, 朝倉書店, (2009).

[6] T. Granlund, et al., The GNU Multiple Precision Arithmetic Library, http://gmplib.org/, 2000-2010.

[7] Y. Hida, X. S. Li and D. H. Bailey, *Technical Report LBNL-46996*, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, 2000.

[8] IEEE, IEEE Standard for Binary Floating-point Arithmetic, ANSI/IEEE Standard 754-2008, IEEE, 2008.

[9] D. E. Knuth, The Art of Computer Programming, Volume 2 Seminumerical Algorithms, Third Edition, Addison Wesley Massachusetts USA, 1997.

[10] C. Lawson, R. J. Hanson, D. Kincaid and F. T. Krogh, *ACM Trans. Math. Softw.*, Vol. 5, 1979, pp. 308-323.

[11] M. Nakata, The MPACK (MBLAS/MLAPACK); multiple-precision arithmetic version of BLAS and LAPACK, http://mplapack.sourceforge.net/, 2008-2010.

[12] J. R. Shewchuk, *Discrete and Computational Geometry*, Vol. 18, 1997, pp. 305-363.