ハイパフォーマンス
コンピューティング 59－7
（1995．12．11）

RISC計算機に適した 3重対角行列の固有値の計算法

日本IBM(株), 東京基礎研究所　寒川　光

1995.12.11

3重対角行列の固有値を求めるアルゴリズムで，スツルム列の性質を用いた2分法が用いられる．スツルム列を計算する漸化式には乗算によるもの以外に，除算による変形も存在し広く用いられている．しかし除算による方法では行列式が求められなくなり，またRISCプロセッサでは除算が遅い機種が多いので，計算速度の観点から好ましくない．

本報告では乗算による漸化式を，RISCプロセッサ向きの標準的なチューニングの手法を適用して高速化する．ここでは2分法に，(算術パイプラインを効率よく使用する目的で) 複数の行列式をまとめて計算する行列式法を加味した手法により高速化し，除算による方法よりも10倍近い計算速度を得た例を報告する．

An Eigenvalue Problem of a Tridiagonal Matrix for RISC processors

Hikaru Samukawa
samukawa@trl.ibm.co.jp
Tokyo Research Labolatry, IBM Japan, Ltd.
1623-14, Shimotsuruma, Yamato-shi, Kanagawa-ken 242, Japan

The bisection method based on the Sturm sequence property is used to solve an eigenvalue problem of a symmetric tridiagonal matrix. The standard recurrsive formulation is often replaced with the divide variant, which seems to be used widly. However, since this variant not only looks over determinant but also the divide operation is relatively slow in RISC processors, it has become quite expensive method for RISC computers.

In this report, some standard ways of programming tuning are applied on the bisection method, in which several determinants are computed together to exploit arithmetic pipelines of RISC processors. This method achieves almost ten times faster than the divide variant.

1 はじめに

最近の RISC プロセッサは，算術パイプラインやスーパースカラー処理方式を実装することで，浮動小数点数の乗算や加算は非常に高速化された．これに対し，除算や条件分岐の処理は相対的にはそれほど高速化されていない．このような RISC プロセッサの計算能力は，最内側ループに並列性が豊かで，if 文を含まないプログラムにおいて発揮されやすい．

対称 3 重対角行列の固有値を，スツルム列の性質を利用した 2 分法で求めるアルゴリズムは，漸化式の依存性のために，RISC プロセッサの計算能力を引出しにくい．またこの漸化式を変形した除算を用いた漸化式を用いると，除算が（相対的に）遅いプロセッサでは，計算は非常にゆっくりしたものにならざるをえない．

本報告では 2 分法に行列式法を加味した方法で，除算を用いる方法よりも 10 倍速い計算速度を得た例を述べる．

2 2 分法による固有値の計算

スツルム列の計算　次数 n の対称 3 重対角行列 T を次のように記述したとき（$b_i \neq 0$ とする），

$$T = \begin{pmatrix} a_1 & b_2 & & \\ b_2 & a_2 & b_3 & \\ & b_3 & a_3 & \\ & & & \ddots \end{pmatrix}$$

スツルム列 $N_s(\omega)$ は，T_i を T の次数 i の主小行列としたとき，$T_i - \omega I$ の行列式を求める多項式 $p_i(\omega) = det(T_i - \omega I)$ を，i が 1 から n まで漸化式の形で計算したときの $p_i(\omega)$ の符号変化の回数によって得られる．

$$\begin{aligned} &p_0(\omega) = 1 \\ &p_1(\omega) = a_1 - \omega \\ &p_i(\omega) = (a_i - \omega)\, p_{i-1}(\omega) - b_i^2\, p_{i-2}(\omega), \quad i = 2, \cdots, n \end{aligned}$$

$p_{i-1}(\omega)$ が求まらないと $p_i(\omega)$ を計算できないという依存性と，$p_i(\omega)$ の符号を調べてスツルム列を数える処理や，$p_i(\omega) = 0$ の場合の処理のために条件分岐の多いループになることが，RISC プロセッサの計算能力を阻害する[1]．

2 分法による固有値の計算　2 分法は，$N_s(\omega)$ が Tの ω よりも小さい固有値の数に一致する性質を利用する．小さい方から k 番目の固有値を求める場合，探索区間 (ω_l, ω_u) で $N_s(\omega_l) < k$, $N_s(\omega_u) \geq k$ が既知とすると，図 1の反復で計算する [1]．

スツルム列を m 回計算することで，区間 (ω_l, ω_u) を 2^{-m}に絞り込める．最初の探索区間はゲルシュゴーリンの定理から，収束判定値 eps は最初の探索区間にユーザーの指定する相対的許容誤差限界を乗じて決めるのが一般的である．IEEE 形式の倍精度浮動小数点数は，仮数部の有効桁が 53 ビットなので，通常は 50 回程度の反復で十分な精度の固有値が得られる．

[1] $p_i(\omega) = 0$ のときは，$p_i(\omega)$ の符号は $p_{i-1}(\omega)$ の符号と等しいとする．

```
10  ω = (ω_l + ω_u)/2
    if(N_s(ω) .ge. k)  then
      ω_u = ω
    else
      ω_l = ω
    endif
    if(abs(ω_l − ω_u) .gt. eps)  go to 10
```

図 1: 2 分法の計算

除算のよるスツルム列計算の漸化式とプログラム　乗算による漸化式では行列式が桁溢れするので，これを変形して除算による漸化式を用いる場合がある．これは $s_i(\omega) = p_i(\omega)/p_{i-1}(\omega)$ とおくことで，$s_i(\omega) = a_i - \omega - b_i^2/(s_{i-1}(\omega))$ と変形される（2 回の乗算が 1 回の除算になる）[2][3]．

```
do i=1,n
 if(ipass.eq.0) then
  s=a(i)-w-bb(i)/s
 else if(ipass.eq.1) then
  ipass=2
 else
  s=a(i)-w
  ipass=0
 endif
 if(ipass.eq.0) then
   if(s.le.0) nsw=nsw+1
   if(abs(s).le.abs(bb(i)*small))  ipass=1
 endif
enddo
```

このプログラムは参考文献 [2] の例を 1 部変更して引用したが，ここでは変数 w に ω が，配列 $a(1:n)$ に a_i が，$b(1:n)$ に b_i が，$bb(1:n)$ に b_i^2 が格納されていて，スツルム列が nsw に得られる．ただし $b(1), bb(1)$ はゼロ，$small$ は許容誤差である．

この方法の利点は桁溢れを気にしないでよいプログラムの単純さにあるが，式が変形されているので，3 重対角行列の特性多項式を計算していることは読み取りにくくなっている．また，(1) 除算が含まれるため，除算そのものに計算時間がかかることと，ゼロ割りを避けるためのチェックが入ることで計算が遅い，(2) スツルム列 $N_s(\omega)$ を数えるだけで行列式を得られない，という難点もあり，利点に比べて失うものも多い．

乗算漸化式　乗算漸化式の桁溢れの問題は，スケーリングによって抑えることができる．この操作は反復数回に 1 回行えばよいので，計算時間に与える影響は小さい．IBM の RISC System/6000[2](RS/6000) で，8 重アンローリングにより 8 回に 1 回スケーリングするプログラムは，除算漸化式によるものより約 3 倍速い．これは RS/6000 では除算は 15 から 20 サイクルを要するのに対し，乗算は 1 ないし 2 サイクルで計算できるからである [4]．

なおスケーリングするので，行列式は $p * 2.0 * *ipwr$ のように指数部の一部を別の変数に分離した形で得られる．

[2]RISC System/6000 は米国 IBM Corp. の商標である．

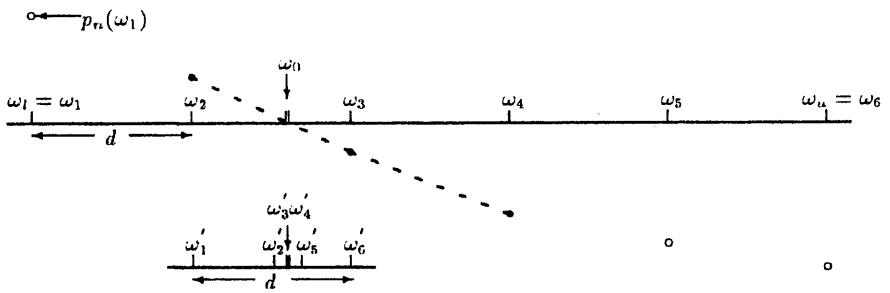

図 2: 5 分法／行列式法

計算時間　対称行列の固有値と固有ベクトルを求める問題では，3 重対角行列に変換するまでがかなりの計算時間を要する．かりに次数 $n = 1000$ とすると，3 重対角化の計算量は $4n^3/3$ なので，200Mflop/s で計算できれば 6.7 秒で完了する．これに対し 3 重対角行列の固有値は，計算量のオーダーが低いので無視されがちである．しかし除算漸化式を用いたプログラムが，スツルム列を 1 つ計算するのに $25n$ クロック周期，固有値 1 つに 50 回程度の反復を要すると仮定すると，100MHz（クロック周期 10 ナノ秒）の計算機では固有値 1 つあたり .0125 秒を必要とする．もし数 100 個の固有値を求めると，3 重対角化の計算時間に匹敵する．

3　RISC 計算機向きの方法

固有値の分離状態　対象区間に**固有値が 1 つしか存在しない**ところまで分離が進んだ状態では，スツルム列 $N_s(\omega)$ を数えなくても，行列式 $p_n(\omega)$ の符号から 2 分法の計算を続けられる（$p_n(\omega)$ が $p_n(\omega_l)$ と同符号なら $N_s(\omega) = N_s(\omega_l)$，異符号なら $N_s(\omega) = N_s(\omega_u)$ と判定できる）．RISC プロセッサは条件分岐の処理が遅いので，この状態では，スツルム列を数えずに行列式だけを求めたほうが速い．

算術パイプラインと多分法　行列式だけを求めるのであれば，漸化式の計算に（スケーリング以外には）条件分岐は現れないので，**複数の ω に対して同時に行列式を計算**することで，算術パイプラインの稼働効率を上げやすい．この効果に着目すると，例えば 3 つの ω に対する行列式が，1 つの行列式を計算する時間の半分以下で求められるのであれば，（m 回の反復で区間を 4^{-m} に絞り込めるので），4 分法のほうが速いということに気付く（パイプラインの稼働効率が 1.5 倍以上になれば，計算量に対する効率が 2/3 になっても速い）．

そこでパイプラインの稼働効率と計算量に対する効率のトレードオフから，同時に求める行列式の数の最適値を求めればよい．この値は一般にパイプラインのもつ “依存性の深さ” から決められる[3]．本稿では段数 2 のパイプラインを 2 本実装する RS/6000 の POWER2 機種を考慮して，行列式を 4 つまとめて計算する 5 分法を用いた．

行列式の利用　固有値の予測値があれば，区間を等分割せずに，存在しそうなところを細かく分割することで，区間をより狭く絞り込める．これには行列式を用いて，特性多項式のゼロ点を予

[3] n 個先の演算命令に影響が及ぶ場合（あるいは次の命令を n クロック周期待たせる場合）を “深さ” n とする [4]．

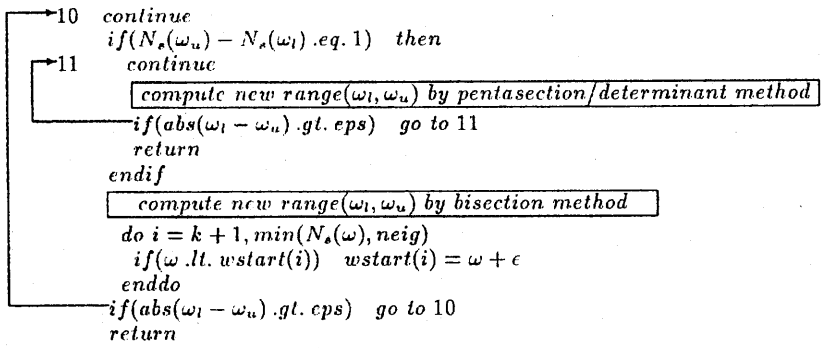

図 3: 2 分法と 5 分法／行列式法

測する**行列式法**（ determinant method ）が利用できる [5][6] .

図 2の上に示したように，$\omega_l = \omega_1$, $\omega_u = \omega_6$ とし，反復の前半で区間 (ω_l, ω_u) を 5 等分する ω_2 から ω_5 について行列式をまとめて求める（ 5 分法）. 固有値を挟む区間と隣接する 1 点からなる 3 点（図では $\omega_2, \omega_3, \omega_4$）を選び，対応する行列式から 2 次補間により固有値の予測値 ω_0 を求める.

反復の後半は図 2の下に示したように，固有値を挟む区間をあらためて ω_1' と ω_6' として，ω_0 を中点とする狭い区間 (ω_3', ω_4') とやや広い区間により ω_2' から ω_5' の 4 点を選び，もう 1 度 4 つの行列式を求める. 区間幅は $d = \omega_6' - \omega_1'$ として，$\omega_4' - \omega_3' = d/64$, $\omega_3' - \omega_2' = \omega_5' - \omega_4' = d/16$ とした（ただし ω_0 が区間端に近い場合はさらに狭くなる）. これらの区間のうちで固有値を挟む区間を次の反復の区間とする. この方法を **5 分法／行列式法**と呼ぶことにする.

もし予測値を含む区間 (ω_3', ω_4') が固有値を挟んでいれば，反復 1 回で区間は 1/320 になる. この場合 8 つの行列式を用いて区間を 1/320 に絞り込んでいるが，$2^{-8} = 1/256$ なので，2 分法でスツルム列を 8 回求めるよりも，スツルム列または行列式を 1 つ求める計算量に対する収束の速さという観点でも効率がよくなる.

2 段階の反復　5 分法／行列式法を導入するため反復を 2 段階に分け，“対象区間に複数の固有値が存在するときはスツルム列を用いる 2 分法を用い，固有値が 1 つに絞り込まれた後は 5 分法／行列式に切り換える”. 図 3にこれを示す. 図では求める固有値の数は $neig$とし，次数 n の行列の小さい方から k 番目の固有値を求める場合を示している. 外側の反復の四角で囲った部分が 2 分法，内側の四角で囲った部分が 5 分法／行列式法になる.

なお 2 分法の反復で求められたスツルム列は，次の固有値を求めるための出発区間を狭くするために配列 $wstart$ に残している [2] . これにより $k+1$ 番目の固有値を計算するための出発区間 (ω_l, ω_u) は，下限が k 番目の固有値，上限が $wstart(k+1)$ になる. 上限はそれまでに計算した $N_s(\omega)$ のうちで，$N_s(\omega)$ が $k+1$ 以上で最も小さい ω である.

実測結果　$n = 1000$ のフランク行列を 3 重対角化した行列に対し，100 個の固有値を求めた[4]. 出発区間を 100 個の固有値に対して固定して，eps をゲルシュゴーリンの定理から得られる出発

[4]次数 n のフランク行列 A は $a_{i,j} = n - max(i,j) + 1$ で正定値である.

区間の $2.\times 10^{-16}$倍とした場合，2分法では固有値1つに平均54回の反復を要した．

本稿の方法では，2分法の反復が平均4回，5分法／行列式法の反復が平均3回に減った．2分法の反復回数が少ないのは出発値を *wstart* に保存した効果であり，出発区間を変更しないと35回かかる．固有値100個を求める時間は，RS/6000 の POWER2 機種の 590 型 (66.5MHz) で0.13秒である．この計算速度は除算漸化式による2分法の約10倍，乗算漸化式による2分法の約3倍にあたる．

この性能は，RISC計算機には苦手なスツルム列を数える処理を，出発区間を狭くすることと，探索区間に1つの固有値しか存在しなくなった時点で行列式法切り換えることによって，できるだけ少なくし，さらに行列式の計算を，複数の行列式をまとめて計算するRISCプロセッサ向きのプログラミングにしたことで達成されている．スツルム列を数える回数が減ったことが性能向上に貢献しているので，重根，あるいは非常に近接した固有値が多く存在すると，性能の向上は少なくなる．

4　おわりに

2分法の計算は，スツルム列を求める漸化式そのものには依存性があるものの，固有値を探索する区間相互には並列性がある．したがって複数の区間について2分法の計算を並列に進める方法が考えられる．ベクトル計算機用のアルゴリズムで，この複数の区間を内側ループに用いてベクトル化する並列多分法が知られている [3]．この場合ベクトル化のために単純なループ・プログラムにする必要性からか，除算漸化式が好まれるようである．ベクトル計算機では乗算に比べて除算はそれほど遅い演算ではないので，これは合理的な方法として使用されてきた．しかしこの除算漸化式による2分法をそのままRISC計算機に持ち込むとかなり不経済ということになる．

本稿では区間ごとの並列性には手をつけずに，**1区間から1固有値を逐次的に求めるアルゴリズム**をチューニングした．この方法は除算漸化式による2分法よりも，RISC計算機では約10倍速く，全固有値を求める場合でも，QR法よりも速いと考えられる．この方法は，並列計算機でプロセッサ数の数倍以上の固有値を求める場合は，区間ごとの並列性で並列化することによって，並列計算機の1プロセッサでほぼこのままの形で稼働するので，有望な方法と考えられる．

参考文献

[1] G. H. Golub and C. F. VanLoan：*Matrix Computations second edition*, The Johns Hopkins University Press, Baltimore and London (1989)．

[2] 森　正武：FORTRAN 77 数値計算プログラミング（増補版），岩波書店 (1987).

[3] 島崎眞昭：計算機科学/ソフトウェア技術講座 9　スーパーコンピュータとプログラミング，共立出版 (1989)．

[4] 寒川　光：RISC超高速化プログラミング技法，共立出版 (1995)．

[5] J. H. Wilkinson：*The Algebraic Eigenvalue Problem*, Clarendon Press, Oxford (1964)．

[6] R. H. MacNeal (editor)：*The NASTRAN Theoretical manual (level 15.0)*, NASA Computer Software Management and Information Center, University of Georgia, Athens, Georgia (1972).