数理解析研究所講究録
第253巻 1975年 38-68

最近の大型計算機の進歩とベンチマークの問題。

日立製作所 神奈川工場　　村田 健郎

〔目次〕

1. はじめに.

最近の大型計算機では，全物的には 仮想メモリ方式，バファアメモリ方式，パイプライン制御，高速乗算機構が採用され，ソフトウェア的には FORTRAN言語仕様の拡張，オブジェクトの最適化機能，リエントラント機能 などが採用されて，ベンチマーク問題についても考慮す可きことが多くなつた。

元来ベンチマークテストは，ユーザー・メーカー相互，或いはもつと狭くメーカー内のハードウエア技術者とソフトウエア技術者との間とか，アプリケーションライブラリ開発者と，ハード・ソフト技術者の間とかの会話の場を提供することによつて，システムの改良 或いは システムの有効利用のための方策を示唆するためのものである。或いは時には，計算機導入に際し，機種選定のために行うこともあるが，その際にも，選定委員と一般ユーザーとの会話の場を提供したり，選定委員と候補メーカー技術者との会話の場を提供したりせねばならない。そう云うわけで，問題の選定，ドキュメントの用意などに相當の気を配らねばその役割を果せないものなので，従来から相当問題であったわけであるが，上述のようなハード・ソフトのすう勢とのからみで，また新たな問題

が持ち上って来ている。今日は、分野を科学技術計算、それも、狭い意味の科学技術計算にしぼって、最近筆者が経験したことを土台にしてこの問題についてお話しして見たい。

2. ベンチマーク問題の選定に際しての基本事項

を要約すると下記のようになろうか。

(1) 目的に応じた ベンチマーク問題の選定が重要である。その際特に、それが如何なる応用分野から、如何なる検討にもとづいて選ばれたものであるか、云い換えれば、その問題の背景にある母集団の輪重及び、その標本としての資格をどの程度持ったものであるかが、関係者に了解可能であることが必要である。

(2) ベンチマーク問題は、その主要アルゴリズムが明示されている必要がある。それは通常の数学的記述が良い。

(3) プログラムは FORTRAN が適当である。その際、評価の対象となっているシステムの FORTRAN 言語仕様と、コンパイラの最適化機能にマッチしたものである必要がある。(マッチしないものは、それが意識的に用意され、特定の目的のための参考データとして使用されることは有って良いが、無意識では評価を誤る。)

(4) ベンチマーク問題としての妥当性の程度が，種々の角度からチエック可能なように，ドキュメントが整備されている必要がある。

3. ベンチマーク問題の選定経過.

現在及び今後の大型計算機の最も重要なテーマは，「今まで不可能であったことを，可能にする。」ことであると筆者は考える。そういう観点から，そのような課題を常に持っている応用分野を，科学技術計算の中から求めると，例えば下記のようなものが具体例としてあがって来る。

(1) 原子力発電施設の安全性解析.

(2) 気象予報のための数値シミュレーション.

(3) 環境保全に関連する種々の拡散過程のシミュレーション

(4) 原子炉設計計算. 例えば中性子輸送コード，拡散コード，燃焼コード，熱水力コード。

(5) 航空機設計計算，安全性解析計算

(6) IC/LSI のデバイスCAD 例えばバイポーラトランジスタの特性解析のための レヨウクレー/ポアソンの拡散モデルの数値解析。

(7) 原子力発電施設等の耐震設計計算。

従来のベンチマークでは，往々にしていきなり上記のような題目に続いてFORTRANプログラムが提示されることがあったけれども，これらの表現は，それぞれの応用分野に従事する科学者・技術者と，それらの管理職，或いは科学技術担当ジャーナリスト・評論家の間の会話の場での表現とでも云う可きもので，それなりの機能を持ってはいるけれども，現在のハードウエア・ソフトウエア技術者のレベルからは余りにも遠い表現である。そこで，そのなかで使われる主要な計算に着目して整理してみると，例えば図1に示されるような系統図に従って，それぞれの専門分野とかゝわり合いながら，結局は同図最下段に示されるような大次元行列の諸解析に帰着される。図1の中段の記述は，繁雑になるのを嫌って，応用上特に重要且つ数値解析的にもよく定式化された所謂自己随伴形の線形の微分作用素関連のものに限って書き込みを行ったけれども，今後不可能を可能にすると云う見地から特に重要な非線形問題，例えば原子炉に於ける中性子や半導体デバイスに於ける正孔や，電子のふるまいを，拡散近似でなくて，そのもとになっているボルツマン形の輸送方程式のまゝで解く問題や，運動媒体中での拡散現象で必ず問題となる非線形連立の偏微分方程式の類も，数値シミュレーションに於ては結局は大次元行列の数値計算に帰着される。

また，線型計画法（LP）或いは多変量解析法，有限要素法と云ったような，既にプログラムプロダクト，或いはアプリケーションパッケージの名で計算機産業の中に定着したものの計算主要部が，やはり大次元行列計算であることは周知であろう。

図1の中段の記述を見ると，これでもクーラン・ヒルベルトの「数理物理学の方法」で扱われているほぼすべての分野をおおっていることも併せ了解されよう。

これで我々の‘母集団の輸重’は可なり明瞭になったと思われるし，‘標本’としては図1下段のマトリックスの数値計算からえらぶ可きであることも了解された。実際の科学技術計算の実務のうえからは，これら方程式を作るまでの謂わば前処理，例えば有限要素法に於ける領域の自動分割などの計算機処理が場合によっては無視できないし，また所謂データーベースを背景にした大規模技術計算に於けるファイル処理の問題もあるが，今日は省略する。

さて，それでは線形代数問題から何をえらぶか．であるが，その方面での現在に於ける標準的テキスト；

Wilkinson - Reinsch ‘Linear Algebra’ (Springer 1971)

からえらぶことにすると，アルゴリズムや適用範囲，数値解

析的注意などの基本的ドキュメントをそれに頼ることができる点など，ベンチマーク問題として大変好都合である。そこで，同書の I/1 ～ I/11 (Linear Systems, Least Squars and Linear Programming) ; II/1 ～ II/18 (The Algebraic Eigenvalue Probrem) の中から，応用上重要且つ，

1°) 基本的アルゴリズムの重複するものは一つで代表させる。

2°) 大次元向きでないものは落す。

3°) 反復法が主要部となるものは反復回数を固定または落す。

といった基本方針で整理して，まず次の7題を選んだ。

I/1 Symmetric Decomposition of a Positive Definite Matrix
(密行列コレスキー法)

I/4 Symmetric Decomposition of Positive Definite Band Matrices
(帯行列コレスキー法．但し SQRT free のもの)

I/6 Solition of Symmetric & Unsymmetric Band Equations
(帯行列クラウト法)

I/10 Singular Value Decomposition and Least Squares Sol.
(特異値分解と最小二乗解)

II/1 The Jacobi Method for Real Symmetric Matrices
(ヤコビー法固有値解析．反復回数固定)

II/2 Householder's Tridiagonalization

II/5 Method of Bisection
(II/2, II/5 を続けて密行列ハウス・バイセクション)

帯行列の固有値解析用として，初の II/7 (QR Algorithm for Band Symmetric Matrices), II/8 (Tridiagonalization of a Symmetric Band Matrix), II/9 (Simultaneous Iteration Method) を検討したが結局これらを採らず，

○ 拡張ハウスホルダ法による帯行列三重対角化法（文献[4]）
○ $Ax=\lambda Bx$ 形の帯に対する Subspace Iteration 法（文献[2]）
○ $Ax=\lambda Bx$ 形の帯に対する Determinant 法 （文献[2]）

を新たに作ることにした。また，有限要素法に於ける Sub-structure 法（ブロック分けの方法）等を重視して，

○ ブロック三重対角コレスキー法

を用意した。また，超大次元の対称密行列のためと，ブロック三重対角コレスキーの主要部が主メモリ容量を超えるような場合（主として三次元問題で起る）を浮きぼりにするためという二つの理由で，

○ 二分割コレスキー法

を追加した。また，上記の諸アルゴリズムを混合して使用する汎用的アプリケーションプログラムの一例として，

◎ 有限要素法による連続体の静的，動的解析プログラム
◎ 熱水力コード COBRA-III

を参考問題とすることにした。

以上は，典型的な線形代数関係であるが，他に注意すべきものに差分法がある。差分法も，不均質，不整形境界の場を扱うときには，有限要素法の場合と同様の連立一次方程式問題に帰するけれども，長方形境界のポアソン方程式或いは，ヘルムホルツ方程式の境界値問題のような，よく整った問題の場合には，連立一次方程式の係数すなわちマトリックスエレメントが各方程式で共通になってしまうものだから，実際のプログラムでは係数をFORTRANステートメントの中に直接書きこんでしまう。この場合は，'長いステートメント'という特徴があるので，コンパイラの能力チェックの意味で一般の連立一次方程式とはちがった面のテストになる。そういうことを考慮して，

○　二次元の場のポアソン方程式境界値問題の差分解法

を一つ追加した。また差分解法の参考問題として

⊙　一層プリミティブモデルの差分解法
（流体力学でいう浅水モデルの類である。二次元のナビアストークス方程式である。）
⊙　中性子輸送コードTWOTRAN

を加えた。（これら差分法は，反復法が主体だから反復回数を固定して行う。）

3' 若干のコメント.

Wilkinson-Reinsch のなかからの取捨選択に関する，若干のコメントを参考までに記して置きたい。

1°) この本では，多くの場合 $\sum_k l_{ik} l_{jk}$ のように，右側の添字がループの内側で動くように書かれている。これは，この本のプログラムのようにAlgol なら丁度良いが，FORTRAN でバッファメモリ方式や仮想メモリ方式の機械を動かすときには丁度具合がわるい方向である。だから，例えばコレスキー分解はこの本のような $A=LL^T$ でなく，$A=U^TU$ だとして計算式をみちびく。と云う具合に直す。

2°) I/10 の Singular Value Decomposition の演算の主役は，非対称行列のQR分解或いはHouseholder 式変換である。このアルゴリズムは，ハウスホルダ法による非対称行列のヘッセンベルグ行列化（II/13. Orthes）や，QR法による帯行列固有値解析（I/8）ハウスホルダ変換による最小二乗解（I/8）の主要アルゴリズムと共通しているので，これらの代表として採用した。

3°) 帯行列に対する固有値解析は，Numer. Math 誌上でこの本の原型プログラムが用意されていた期間（1965~1970）のあと，有限要素法関連の固有値解析のため特に重視されて，

重要な発展を遂げた。II/7, II/8, II/9 を検討したがこれを採らずに，文献[4]，[2]によって別途用意したのはそのような理由による。しかし，II/8 は兎も角として，II/7（帯に対する直接QR）や II/9（Simultanious Iteration）は相変らず学ぶ可き多くのものを持った立派なプログラムであることに変りはない。尚ほ，帯の $Ax=\lambda Bx$ 形に対して，ランチョス法系の新方法が Stanford 大學の Golub, Underwood, Wilkinson* によってなされたと傳えられるが，（文献[3]）それは筆者は未だ見る機会を得ないが注目す可きものと思われる。

4°) II/1 の Jacobi 法は大次元には向かないし，また反復法である点でもベンチマーク向きでないけれども，あまりにも有名だから反復回数を固定して採用した。

5°) 対称疎行列の固有値解析のための II/2 ハウスホルダ三重対角法は異存のないところであるが，そのあと，Bisection-Inverse Iteration と続けるとか，QR-Inverse Iteration と続けるかしないと答えが出ないわけである。所がそれらは反復法のため，データ依存性があってベンチマーク向きでない。ここでは，実用上三重対角化が最も重要であると云う観点から，II/2 だけを問題にすることにし，答えのチエックを速く見ると云う便宜のため，II/5 Bisection をつなげた。（固有値を全部プリントするのは嫌だから，最大，最小から夫々20箇。）

---

* J.H. Wilkinson が Stanford 大学の客員教授になっている。

4. 標準ベンチマーク問題の説明と具体的問題点.

（抜粋）

4.1 コレスキー法による連立一次方程式解法.*

● 背景

対称正定値行列は，非対称と比べて実用上はるかに使用頻度が高い。特に，帯行列或いはブロック帯行列が重要である。図1の中段を見よう。我々が最も多く取扱う所の，

$$-div(K\,grad\,u)+du=f \qquad \text{（平衡問題）}$$

或いは，

$$-div(K\,grad\,u)+du=\lambda u \qquad \text{（固有値問題）}$$

は，数学の方では「自己随伴楕円形方程式の境界値問題，或いは固有値問題と呼ばれ，これらを離散化して得られる行列は，差分法でも有限要素法でも対称正定値の帯行列である。有限要素法と仮想メモリ方式の普及に伴って，相当広巾の帯も扱うようになったから，密行列についてもそれが広巾の帯を扱うに際しての問題点を浮きぼりにすると云う意味で，密行列のテストデータに関心を持つ必要が出て来るが，第一の関心は，帯及びブロック帯に向うだろう。

＊ ベンチマーク問題に附けるドキュメントとしてどれぐらいのものを用意すれば，関係者一同が「はは、なるほど」と了解するかの一例を示すつもりでやや詳しく書いた。

● アルゴリズム及びプログラム上の注意

U^TU 分解と，Squar Route free の U^TDU 分解とがある。密行列に対してはどちらでも良いが，帯に対しては U^TDU 分解による可きである。内積計算部はいつでも二倍長でやる可きである。対称性を積極的に利用するプログラムを標準とす可きである。U^TU 或いは U^TDU 分解及びその次のステップの所謂前進代入；（例えば U^TU なら $U^Tz=b$）は普通にプログラムすれば自然に局所性の良いプログラムができるが，後退代入；（$Ux=z$）の方は注意が要る。一つのベクトル b に対して，$U^Tz=b$，$Ux=z$ を一回解くだけの場合には，コレスキ分解の方が主要部だからそれほどでないが，コレスキー分解一回に対して，前進代入，後退代入を多数回繰返すとき特に問題になる。

○ $A=U^TU$ 分解のアルゴリズム；$A=(a_{ij})$，$U=(u_{ij})$ とするとき，$a_{ij}=\sum_{k=1}^{i-1} u_{ki}u_{kj}$ の関係から次式を得る。

$i=1$ のときは特別に for $j=1,n$ $u_{1j}=a_{1j}/u_{11}$ で始末して，

for $i=2,n$ {
$u_{ii}=SQRT\left(a_{ii}-\sum_{K=1}^{i-1}u_{Ki}^2\right)$
for $j=i+1,n$
$u_{ij}=\left(a_{ij}-\sum_{K=1}^{i-1}u_{Ki}u_{Kj}\right)/u_{ii}$

（実際には u_{ij} は a_{ij} のうえに重ねる。）

○ $U^T z = c$ の解 z を求めるアルゴリズム　$c = (c_i)$ として

for $i = 1, n$　$z_i = (c_i - \sum_{k=1}^{i-1} u_{ki} z_k) / u_{ii}$

○ $Ux = z$ の解 x を求めるアルゴリズム．これが問題．

$x_n = z_n / u_{nn}$

for $i = n-1, 1$　$x_i = (z_i - \sum_{k=i+1}^{n} u_{ik} z_k) / u_{ii}$

では，プログラムの局所性がわるいので次のようにやる。

$x_n = z_n / u_{nn}$

for $i = n-1, 1$ {
　for $j = 1, i$　$z_j = z_j - u_{ji} z_i$
　$x_i = z_i / u_{ii}$

ここに一次元アレイ (z_j) は二倍長にとる。

○ 注意　$U^T U$ 分解アルゴリズムは次のようにもやれる。

$r_1 = 1 / SQRT(a_{11})$

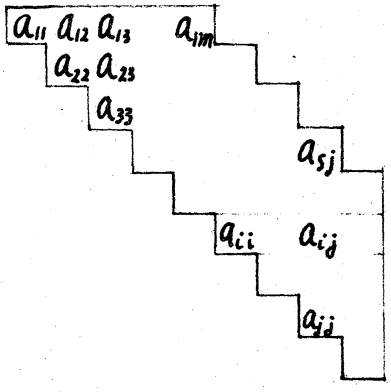
for $j = 2, n$ {
　for $i = 1, j-1$
　　$u_{ij} = (a_{ij} - \sum_{k=1}^{i-1} u_{ki} u_{kj}) * r_i$
　$r_j = 1 / SQRT(a_{jj} - \sum_{k=1}^{j-1} u_{kj}^2)$

$A = (a_{ij})$ の上半分を $a_{11}, a_{12}, a_{22}, a_{13}, a_{23}, a_{33}$, の順に，長さ $n(n+1)/2$ の一次元アレイに收容してプログラムするわけである。そこで，$n(n+1)/2$ が主メモリ容量を超えないときは，

初めのものでも，ここで示したものでもどちらでも良いが，$n(n+1)/2$ が主メモリ容量を大巾に超えるときには，応急処置が必要になる。そこまで考えると，むしろこの注意の所で述べたアルゴリズムに従ってプログラムして置いた方が良いと思われる。

○ 帯行列 $A = U^T D U$ 分解アルゴリズム

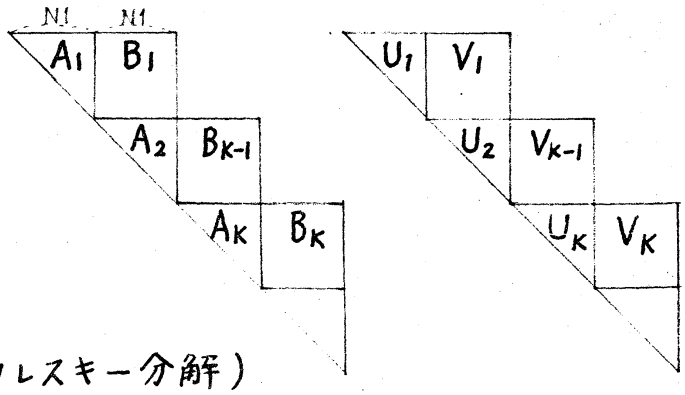
$i \leqq j \leqq i+m-1$ 即ち帯半巾を m とする。



for $j = 1, n$

$$\begin{cases} \text{for } i = s, j \quad \tilde{u}_{ij} = a_{ij} - \sum_{k=s}^{i-1} u_{ki}\tilde{u}_{kj} \\ \text{for } i = s, j-1 \begin{cases} u_{ij} = \tilde{u}_{ij} \cdot r_i \\ s = s + \tilde{u}_{ij} u_{ij} \end{cases} \\ d_j = a_{jj} - s \\ r_j = 1/d_j \end{cases}$$

○ ブロック三重対角行列 $U^T U$ 分解のアルゴリズム



$$\begin{cases} A_K = V_{K-1}^T V_{K-1} + U_K^T U_K \\ B_K = U_K^T V_K \end{cases}$$

より，

$$\begin{cases} A_K \underset{(新)}{=} A_K \underset{(旧)}{-} V_{K-1}^T V_{K-1} \\ A_K = U_K^T U_K \quad (A_K \text{をコレスキー分解}) \\ B_K = U_K^T V_K \quad (V_K \text{を求めて} B_K \text{のうえに重ねる}) \end{cases}$$

ここで実例によって，

1°) バッファメモリ方式／仮想メモリ方式

2°) FORTRAN言語仕様とコンパイラの最適化機能

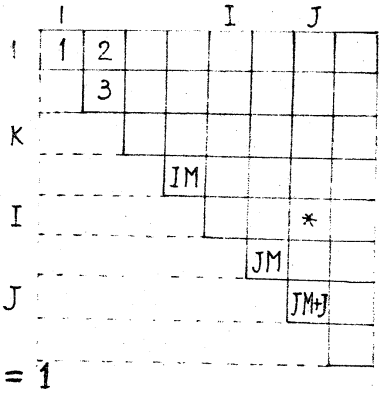
3°) パイプライン制御，高速演算機構

に適合したプログラムとはどんなもので，不都合なプログラムはどんなものかを説明しよう。例題としては，U^TU分解の主要部；

$$\text{for } i=1,j-1 \quad u_{ij}=(a_{ij}-\sum_{k=1}^{i-1} u_{ki}u_{kj})*r_i$$

を採る。一番初めに示すA）プログラムが好ましいものであり，B）C）は何等かの意味で，2°) 3°)の機能を生かせないプログラムになっている。

A)
```
      DO 20 I=1,J-1
      IM =IM+J-1
      SM = 0.0 DO
      DO 10 K=1,I-1
      SM = SM+A(IM+K)*A(JM+K)
   10 CONTINUE
      A(JM+I)=(A(JM+I)-SM)*R(I)
   20 CONTINUE
```



B)
```
      JM=JM+J-1
      J1 =J-1
      DO 20 I=1,J1
      SM=0.0DO
      I1 =I-1
      DO 10 K=1,I1
      JK = JM+K
      IK = IM+K
      SM = SM+A(IK)*A(JK)
   10 CONTINUE
      JI =JM+I
   20 A(JI)=(A(JI)-SM)*R(I)
```

C)
```
      I = 1
   20 SM = 0.0 DO
      IK =IM
      JK =JM
      K=0
   10 IK =IK+1
      JK =JK+1
      SM = SM+A(IK)*A(JK)
      K=K+1
      IF(K.LT.I) GO TO 10
      JI = JM+I
      A(JI)=(A(JI)-SM)*R(I)
      I = I+1
      IF(I.LT.J) GO TO 20
```

1° J(J+1)/2 が主メモリ容量Mを超えると，急激にドラム参照のためのアイドルタイムが増す。そこで応急処置としては，$J^2/2 \leqq M$ のうちはこのプログラム通りやり，$J^2/2 > M$ になつたら，初めに示したアルゴリズムに基くプログラムに切換える。DIMENSIONのとり直しをしないで局部的なプログラムの書き直しで済すと云う意味で応急処置と言った。二分割コレスキー法によるのがオーソドックスであることは言うまでもない。

2° FORTRAN言語仕様の狭いものでは，B)またはC)のようなプログラムになる。最近の大形計算機用のFORTRANでは大低A)のように書ける。そしてA)だと最適化機能が働いて，DO 10 の中のステートメントは5～6命令でやってしまうが，B)C)だと最適化の程度が弱く，12～15命令になってしまう。

3° 乗算の遅い機械だとA)も，B)C)も乗算の回数が同じため大差ないのだが，2倍精度乗算がストア命令並みにまで高速化された機械*だと大差が出る。（8800 では 2~2.5倍）

ついでながら，従来，対称行列の場合にもかゝわらず，上記のように一次元アレイでプログラムしないで，二次元アレイでやった方が，主メモリに入りさえすればその方が速い。と云うことがあったのも上の説明で判ったであろう。新しいFORTRANではもうそのようなことはないから断然一次元アレ

*例えば 360/195 , HITAC 8800

イによる可きである。

4.2 密行列固有値解析

3'若干のコメントの項でもふれたように，Jacobi 法は大次元向きでない。実際 II/1 のアルゴリズムは実によく考えられた優れたものであるにもかゝわらず，2倍精度100元の固有値解析に HITAC 8800 をもつてしても 35 秒を要する。(反復回数13回で。) Householder - Bisection - Inverse Iteration 或いは Householder - QR - Inverse Iteration では大体 3～5 秒で同じ精度が得られるから，100 元ですでに一桁余計に時間を費すわけである。Householder 法系のものも，初めの三重対角化の部分だけを正規のベンチマーク問題とする。理由は，後半は反復法のため入力データ依存性その他ベンチマーク問題としてはあまりにも考慮す可きことが多すぎることと，Inverse Iteration の主要部は三重対角行列の連立一次方程式解法であるという意味で，別途テストされるからである。3'のところでものべたように，我々は，Householder 法のところを計時するようにプログラムし，たゞチエックの意味で Bisection 法で，最大から20箇，最小から20箇の固有値を計算させてプリントすることにしている。(500元もものマトリックスの固有値を全部打ち出すのはかなわない。)

○ ハウスホルダー三重対角化法のプログラム上の注意.

ハウスホルダー三重対角化法は周知のように,

$A_{i+1} = P_i A_i P_i$ （ここに $P_i = 1 - u_i u_i^T / h_i$ ）

と云う変換を,

$$(1 - uu^T/h) A (1 - uu^T/h) = A - uq^T - qu^T$$

の形に直して行う。これを次の順にプログラムする。

$$p = Au/h \quad \text{----------(1)}$$

$$h = u^T p / 2h \quad \text{----------(2)}$$

$$q = p - hu \quad \text{----------(3)}$$

$$A = A - uq^T - qu^T \quad \text{------(4)}$$

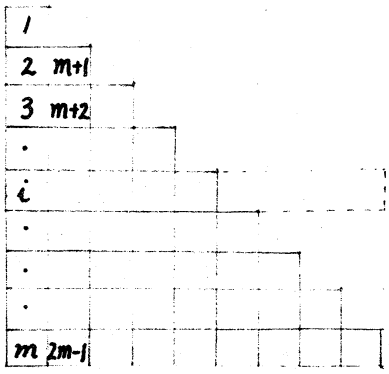
ここに，計算の主要部すなわち $m(m+1)/2$ 箇の要素の計算をするのは(1)と(4)である。プログラム上の注意事項は二つあって,

1°) (1)の一次変換 Au そのもの

2°) (1)に於けるAの参照順序と(4)に於けるAの参照順序を丁度逆方向に行ってプログラムの局所性をよくすること。

の二点である。対称行列は上半分または下半分だけを一次元アレイに記憶してやるのが推奨される。そのとき一次変換をどうするかは，この問題に限らずしばしばあらわれるから，ここで詳しく見て置きたい。

今，$A=(a_{ij})$の下半分を右図のような順序に番号づけられた一次元アレイに記憶しているとする。しかし簡単のため以下の算式では a_{ij} は二次元アレイ式のそのまゝの表現をとる。



常識通り，一次変換 $\tilde{w}=Aw$ を，

$$\tilde{w}_i=\sum_{k=1}^{i-1}a_{ik}w_k+\sum_{k=i}^{m}a_{ki}w_k$$

のまゝにプログラムすると，右辺の初めの項の部分の局所性を損う。そこで次のアルゴリズムに従う。

($v_j=0$ 但し $j=1,2,\cdots,m$ からスタートして，)

for $i=1,m$ {
  $s=v_i$
  for $k=i,m$ { $s=s+a_{ki}w_k$; $v_k=v_k+a_{ki}w_i$ }
  $\tilde{w}_i=s$
}

4.3 帯行列固有値解析

拡張ハウスホルダー法による三重対角化を経由する方法をベンチマーク問題の標準としてとり上げた。この方法は丁度一年前のこの集会で発表した方法のうちの素直な方法の方である。（文献4の図4-3，図4-4に示した方法。）この方法

は，$Ax=\lambda x$ 形専用であるが，その限りに於ては，必要固有値が多いとき，特に強力である。$Ax=\lambda x$ 形に限る場合の Subspace Iteration 法や，Determinant 法とのなわばりは凡そ図5の如くである。但しこれらの方法については Bathe・Wilson（文献[2]）に従った。これらの方法は反復法共通の欠点として当然のことながら，入力データ依存性がいちぢるしいので，元来ベンチマーク向きでないが，拡張ハウスホルダ法と云う新しい方法の位置づけを行うためにはこれらがよく知られているだけに好都合である。図2では，Sub-Space Iteration は反復8回，Determinant 法は反復6回として作図をしておいた。これら二つの方法は，アルゴリズムの主要部は，U^TDU コレスキー法や，行列積の計算だから，純すいにベンチマークだけの目的からするとやる必要のないものである。

ベンチマーク問題ということを離れて実用上の観点から考えて，$Ax=\lambda Bx$（A，B 共に対称帯）について Determinant 法や Subspace Iteration 単独で攻めるのは，与えられた行列の性質についての予備的な知見がない場合には危険を伴う。そこで，予じめ‘集中質量近似の方程式 $Ax=\lambda Dx$（D は対角行列）を拡張ハウスホルダ法で解いて，それによって得られた固有値，或いは固有ベクトルを，Determinant

法或いは Subspace Iteration の初期値にして，これらの方法で Iterative Refinement を行う，ということによって，これらの方法の実用範囲を大巾にひろげられるように思うが如何なものか？。その場合の反復回数は2回とか，高々4回以下で打ち切りで良いと思われる。

5. おわりに.

仮想メモリ方式・バッファメモリ方式，パイプライン制御，高速乗算機構といった最近の大形計算機ハードウエアの傾向と，FORTRAN言語仕様の拡張，コンパイラに於けるオブジェクトの最適化機能といったソフトウエア的な傾向にそくしたベンチマーク問題はいかに有る可きかの問題をとり上げ，ついでにこれら傾向にマッチしたアルゴリズム，及びプログラム技法についてお話しした。こう云う話は，具体的にしないと話が発散し勝なので，実際の問題を例にとって述べて見たわけである。併せて，ベンチマーク問題を云々する場合には，それが如何なる分野の如何なる意味での代表であるかを明うかにする必要があると云うことを強調し度いがために，いさゝか蛇足めいた話もした。ここで，紙面の関係で言い足りなかったことを含めて，要約めいたことを行うとすると，およそ下記の如くなろうか？。

1° これから狭い意味での科学技術計算能力は，ハード的にも，ソフト的にも，従来の方法ではそう目覚ましい進歩は望めなくなっている。ベクトル演算機構とか，アレイプロセッサを作るというような方法でも，そう多くは望めない。特に，FORTRAN とか PL/1 のワク外のプログラミング努力を要する行き方は，金物の進歩よりもはるかに可能性の多い数値計算技法の進歩に順応して行くのに不都合であるので，金物は，FORTRAN 或いは PL/1 で完全にその能力を使い切れるものでなければならない。そうした場合，FORTRAN 等の言語仕様とコンパイラの最適化の可能性とのからみで金物を考えねばならない。そのためには，アプリケーションの動向を標本化したベンチマーク問題の設定，及びそれによるデザイレビューが基本的に必要である。

2° その役割をはたす可きベンチマーク問題は，使う側と作る側との討論の接点であり，今後の応用分野，数値解析の動向などまでふまえた大テーマであるから，学会のような，権威ある人々と場に於てとりあげていただきたいと思う。

3° 筆者のようなメーカー人ではどうしても視野が狭くなってビジョンに欠けるが，今回の試みから判ったことを列挙すると以下の如くである。

3.1° 最近のコンパイラによれば*，その性質についてのマ

* 今回主として調べたのは 360/370 FORTRAN H opt 2, Extended 及び，HITAC 8700/8800 OS-7 FORTRAN OPT-2 である。

クロな知識さえあれば，行列関連の標準的問題に対して，専門的なアセンブラープログラマ並みのよいオブジェクトを出すようなFORTRANプログラムを容易に書くことが出来る。

3.2° 一方，昔ながらのFORTRANの知識で書かれたプログラムでは，たゞ単に金物が速くなった分だけの利益を得るに止り，本当の能力の半分も引き出し得ない。

3.3° 従って，ベンチマーク問題は勿論のこと，ライブラリープログラムや，一般のプログラムに於ても，新しいFORTRANに即したプログラム技法の見直しが必要である。と云っても要点は簡単である。はじめに挙げたバッファメモリ方式等金物の特徴を上手に使うことも含めて，凡そ次のことぐらいに気をつければ良い。

a) アレイの添字の動かし方。一番内側のループで一番左側の添字を動かす。

b) DO文の制御変数はループの中で置き換えしない。ループ中の主要ステートメント内にexplicite(陽)に書かれてあるようにする。例えば，

```
      DO 10  K = 1, N
      SUM = SUM + A(I+K-1,J) * B(N-K+1)
   10 CONTINUE
```

以上二つが特に重要であるがついでにつけ加えると，

c) DO文の中で，或る添字の値（多くの場合 K=1とか，K=N)の場合特別なことをやらせる必要上 IF文を使うが，それはできるだけさけて，その処置は DOの外でやる。

d) DO文の中にIF文があり，その行き先が DOループの外であるときには特に時間がかゝることを知る。

e) 一番内側の DO文の中では，制御変数に限らず一般にSTORE命令を無駄に増すような変数の置きかえをやらない。

f) DO文でまわせるループを IF文でまわすとおそい。

3.4° 仮想メモリ方式は，上に述べた 3.3° a) の注意さえ怠らずにやれば，帯行列に対しては実に具合良く機能する。密行列に対しては，ブロック化手法と云う若干技巧的なプログラム技法を要することがある。（例えばコレスキー法）それをやらないで済ますためには，平均命令実行時間に対してバックアップメモリ（ドラム）のアクセスタイムが 10^4倍以下，欲を云えば 2×10^3倍以下であって欲しい。でないとすると，帯行列でも，今後空間三次元の場の問題を有限要素法等で本格的に相手にするようになると，やはりブロック化による局所性の向上，或いはブロック反復法のような準直接法が必要である。

〔文献〕

[1] Wilkinson · Reinsch Handbook for Automatic Computation Vol.II. Linear Algebra (1971) Springer-Verlag

[2] Bathe · Wilson 'Solution Method for Eigenvalue Probrems ...' (1973) Int.J. for Num. Meth in Engng. Vol6

[3] Golub · Underwood · Wilkinson 'The Lanczos algorithm for the symmetric $Ax=\lambda Bx$ Probrem (1972) Tech. Rep. No. CS270 Stanford Univ.

[4] 村田 '仮想記憶を意識した大次元固有値解析 (1973) 京大数理解析研究所講究録 第199号 (1974年1月)

以下に示す図のうち，図2，図3-1，図3-2，図4のデータは，本文に示したアルゴリズムに従ったFORTRANプログラムをHITAC 8700/8800用OS-7 FORTRANコンパイラによってコンパイルしたオブジェクトコードを，工業技術院大型プロジェクトによって試作された「超高性能電子計算機」によって測定したものである。この機械の平均命令実行時間は約250nSドラムの平均アクセスタイムは約10mS，転送レート2mS/4KBである。

図2，図3-1，図3-2に於ては，夫々のプログラムに対して，650KBの主メモリワクをとって計算し，図4に於ては，主メモリワク650KBの場合と1650KBの場合の両方が示してある。また，図2，図4に於ては，ドラムアクセスタイムが2.5mS，転送レート0.5mS/4KBの場合も概算によってP付記してある。図4で判ることであるが，主メモリワクが大きい場合の方が，$(T_U-T_C)/T_C$ すなわち Idle Time/CPU Time が小さくなる。

図1. 各種応用問題と解法の系統図の一例

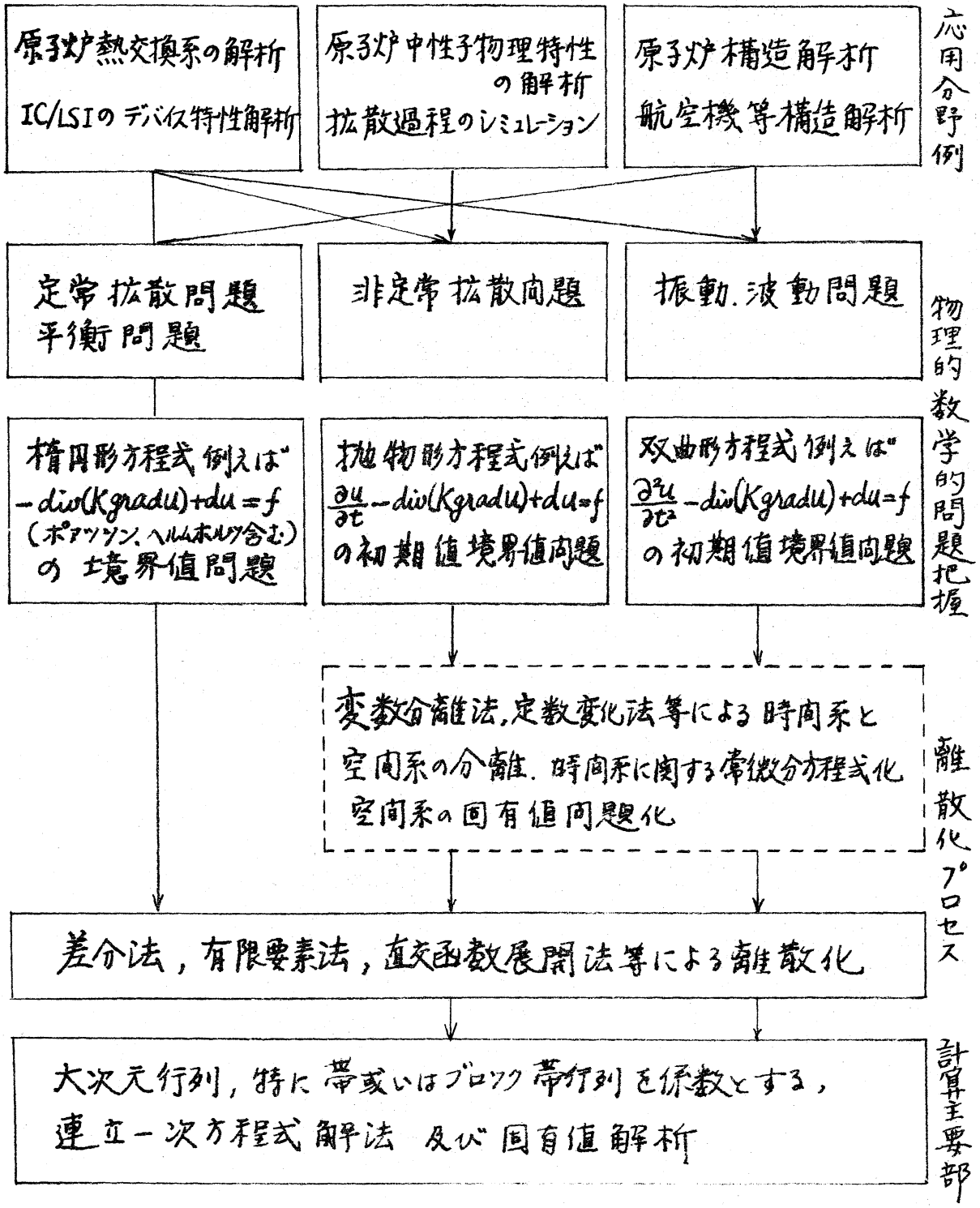
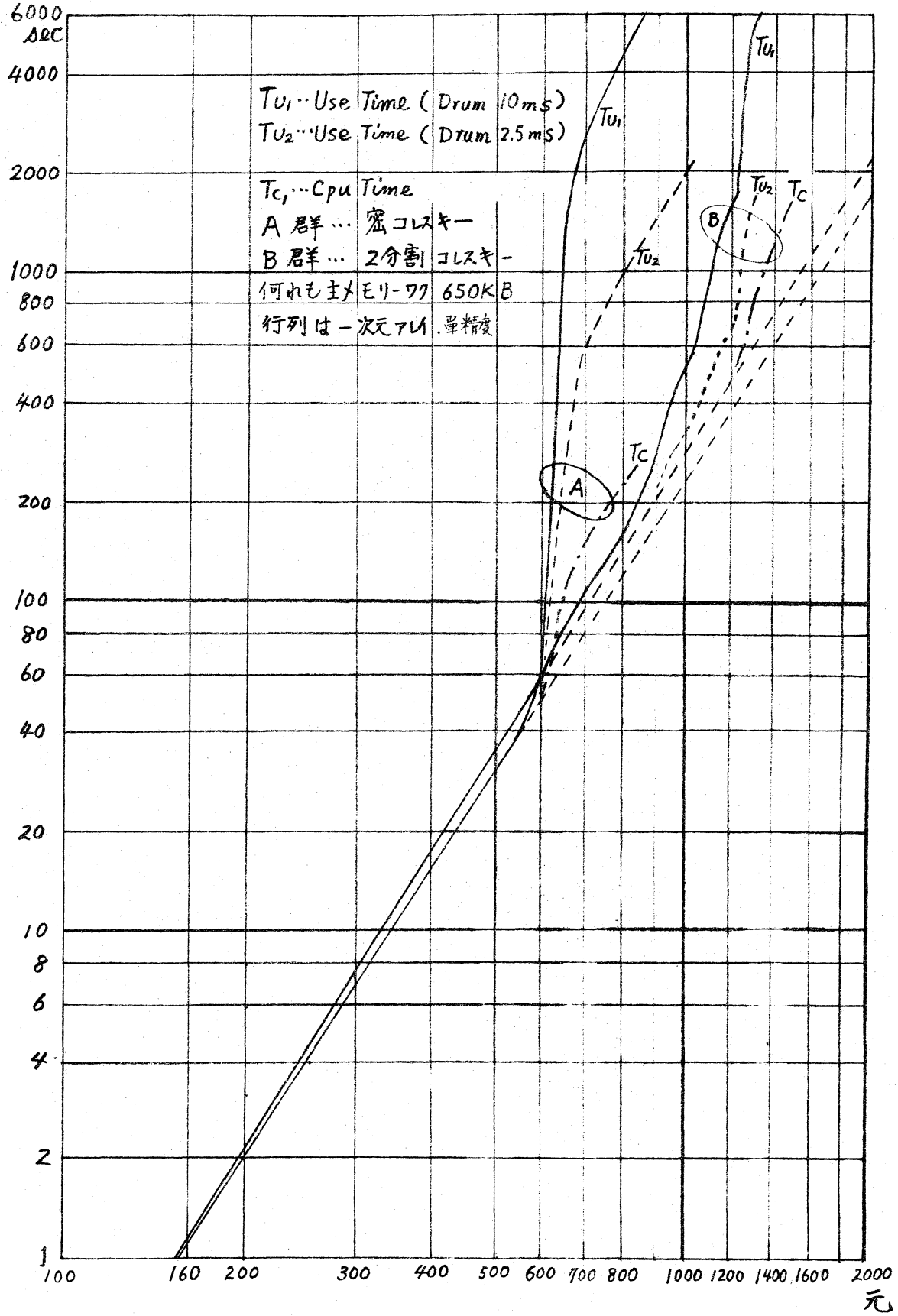


図2. コレスキー法による連立一次方程式解



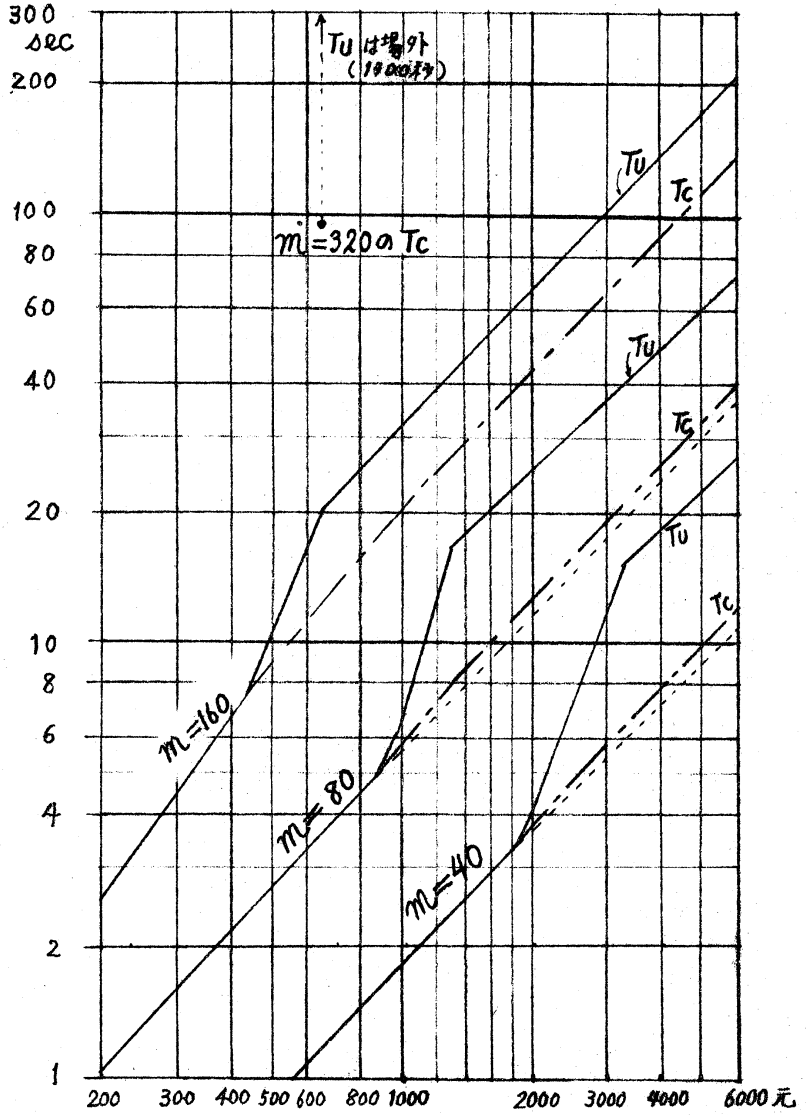


図3-1 帯行列UTDU分解(主メモリワク650KB)

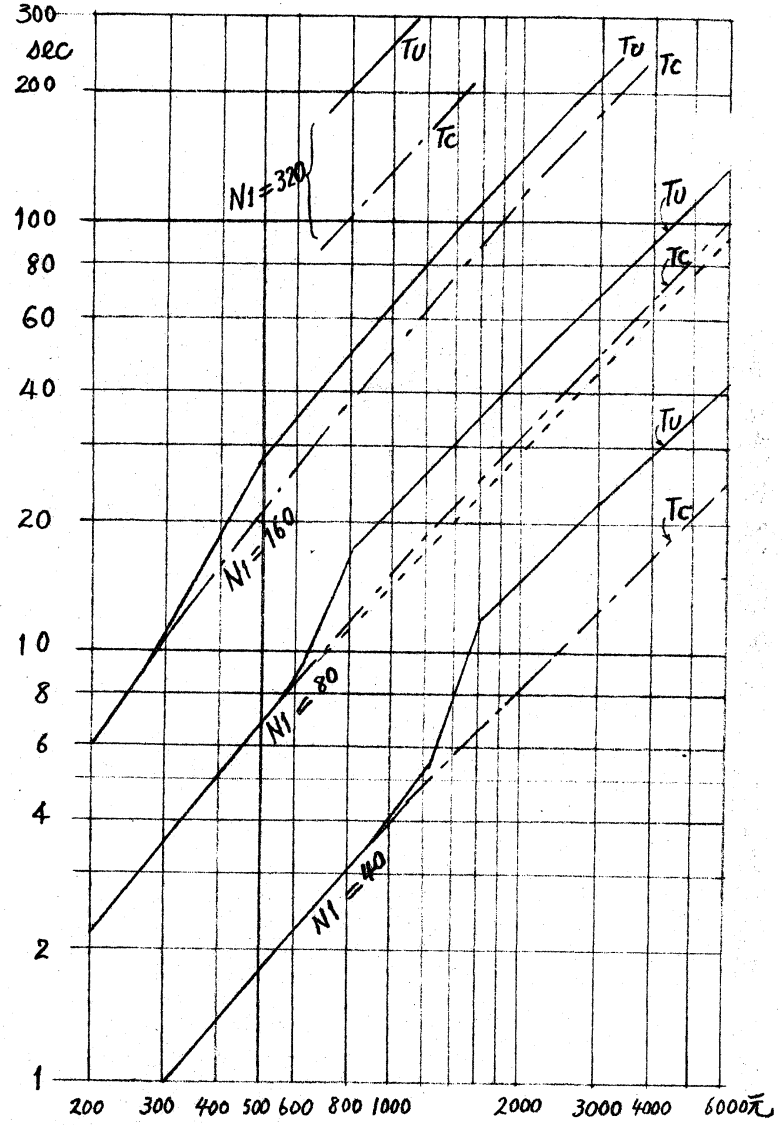
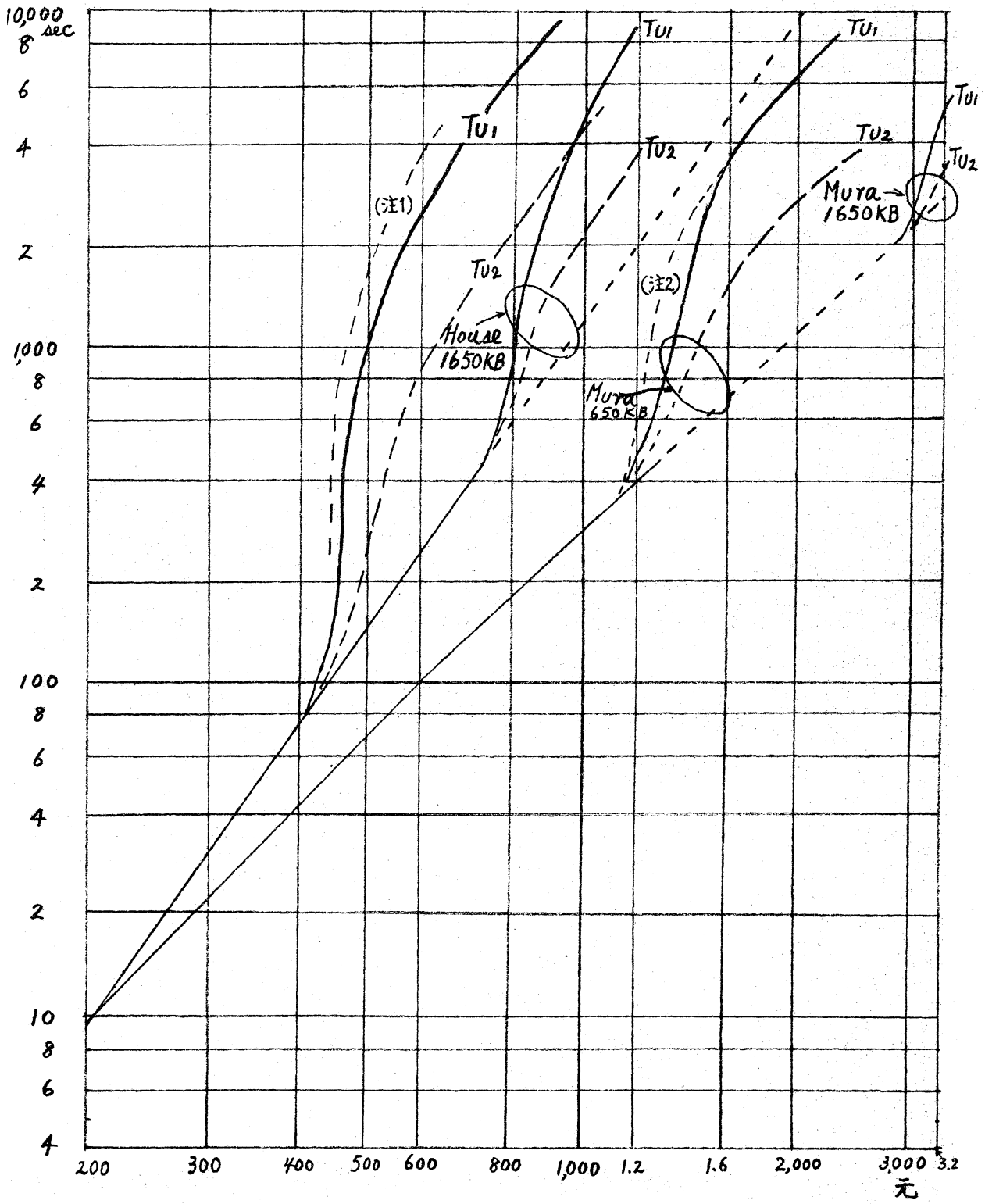


図3-2 ブロック三重対角行列(小行列の元数N1)のUTDU分解(主メモリワク650KB)

図4. ハウスホルダー法と拡張ハウスホルダ法（帯半巾 $m=40$ の場合）
（行列要素 2倍精度）

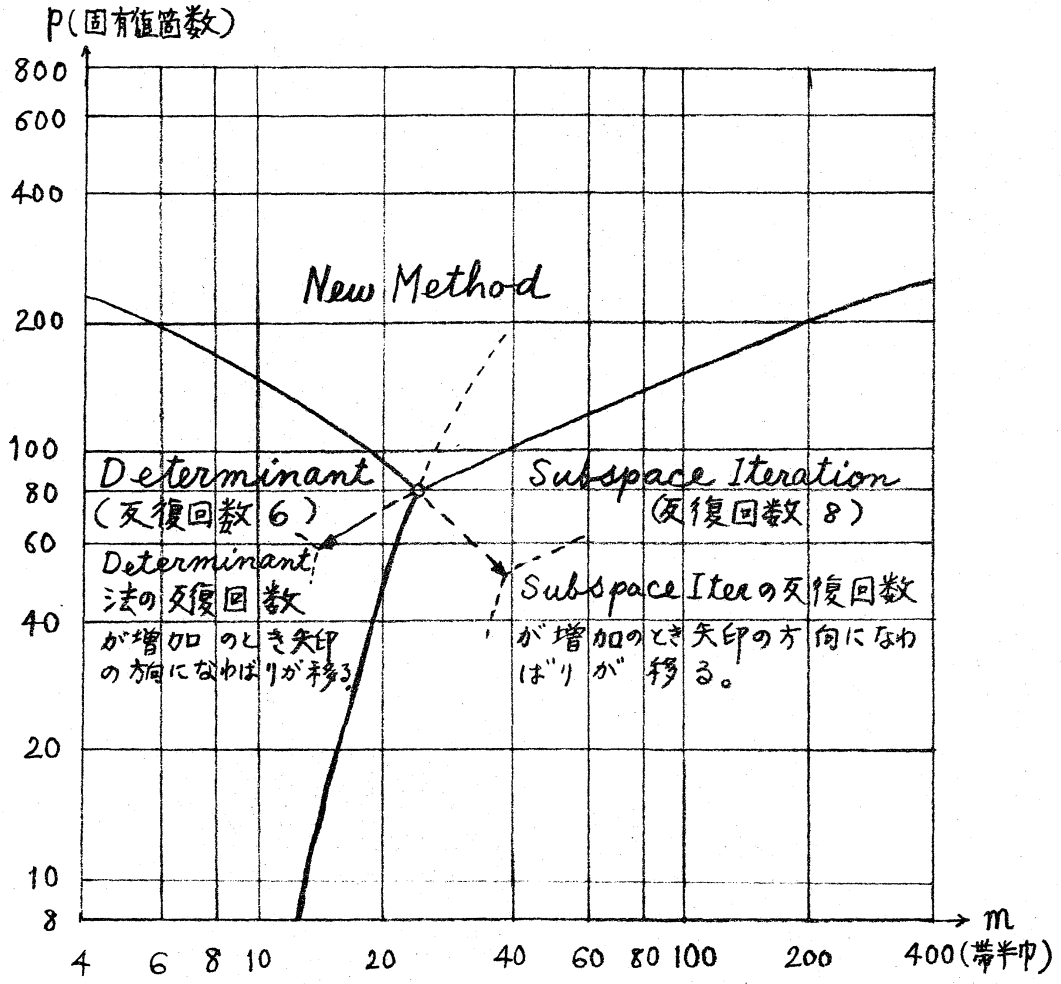


（注1）この実線は $p=Au/h$（往），$A=A-uq^T-qU^T$（往）の場合

（注2）この実線は，文献[4]の図4-1，図4-2の方法 つまり一番簡単な場合．

図5. 対称帯行列固有値解析（$Ax=\lambda x$ 形）諸法の適用領域図〔元数4000に固定，縦軸は固有値個数p，横軸は帯半巾mである。それぞれの所要演算回数は次式で概算〕

- New Method　$2n^2m$
- Subspace Iter.　$2\beta nq(m+q)$，βは反復回数，$q=1.25p$
- Determinant　$\frac{1}{2}\beta nm^2p$，βは一固有値当り平均反復回数



〔注意〕以上の考察には，必要主メモリ容量についての考慮が入っていない。それを考慮に入れると，Determinant法が mn 語であるのに比べて，New Method（拡張ハウスホルダー法）は $2mn$ 語である。また，Sub-Space Iteration は $2nq+q^2$ 語で，q すなわち必要初期ベクトルの本数に依存する。これらが主メモリ容量を超えるときのアイドルタイムはそれぞれの算法独得で，平均命令実行時間にたいしてドラムのアクセスタイムがどれぐらいの倍数であるかにも大いに依存するので，比較は容易でない。