TR-AC-0026

012

行列の固有値と固有ベクトルの近似解法

野口 孝明　新上 和正

1999. 2. 4

ATR環境適応通信研究所

行列の固有値と固有ベクトルの近似解法

野口 孝明, 新上 和正

概論

対象とする系 (= 行列) が時間発展を遂げており, 系の変動が比較的小さい時に各時間ステップにおける固有値・固有ベクトルを高速に求める場合において, 共有メモリー型ベクトル並列計算機上で並列処理機能とベクトル処理機能を有効に生かすことの出来る近似計算アルゴリズム (SD:Simulated Diagonalization) を提案し, 検討する. 実際に行列 $F(t)$ が時間発展を遂げている場合に適用した結果について報告する.

目次

本報告書の構成

- 第 1 章 対称行列の固有値と固有ベクトルの近似解法
  対称行列の固有値と固有ベクトルを従来の三角行列を経由せずに, 主として線型計算で構成された固有値解法を提案する. これは, 高いベクトル・並列処理性能を示す.

- 第 2 章 時間変動する系への適用
  本手法を時間発展する行列に適用し, 本手法の有効性を検討する. 既存ライブラリにある固有値ルーチンとの比較において議論する.

- 第 3 章 準 Newton 法の適用
  最適化問題に帰着された固有値問題に対して, 非線形最適化手法を適用する. 2 章で適用した高次元化アルゴリズムとの比較において議論する.

第 1 章

対称行列の固有値と固有ベクトルの近似解法

1.1 従来の固有値解法

対称行列の固有値は,Householder 変換で三重対角行列に変換し, その後すべての固有値が必要なら QR 法, 一部の固有値が必要なら二分法を利用するのが一般的である. 固有ベクトルについては, 全固有ベクトルが必要なら QR 法の中で別計算を行い, 一部のみでよいときは, 逆反復法で求めることができる.[1][2][3] 並列処理に関しては,Householder 法は, 三重対角化の処理が進むにつれて対象となる行列の次数が下がるため同時に計算できる要素が減少し並列処理には向かないと言われていたが,[4] 近年分散メモリ型並列計算機上での性能評価が報告されている.[5][6] 他方,Lanczos 法のアルゴリズムは線形計算で構成されているため並列処理に向いているが, 丸め誤差の累積に弱く得られた固有ベクトルが直交しないことが課題となっている.[4]

1.2 固有値問題における摂動

一般に固有値問題に与える摂動の影響は,A の固有値 λ_1 に対して摂動 $B(=(\beta_{ij}))$ を与えると, $A+\varepsilon B$ の固有値 $\lambda_1(\varepsilon)$ は,

$$\lambda_1(\varepsilon) = \lambda_1 + \varepsilon\beta_{11} + \frac{\varepsilon^2}{k}\sum_{j=2}^{n}|\beta_{1j}| \tag{1.1}$$

となるため,ε が充分小さければ, 変動も小さいことが期待される.
一方, 固有ベクトルに関しては,A の固有ベクトル $\mathbf{x}_1$ に対して, $A+\varepsilon B$ の固有ベクトル $\mathbf{x}_1(\varepsilon)$ は,

$$\mathbf{x}_1(\varepsilon) \cong \mathbf{x}_1 + \varepsilon\left\{\frac{\mathbf{x}_2^T B\mathbf{x}_1}{\lambda_1-\lambda_2}\mathbf{x}_2 + \cdots + \frac{\mathbf{x}_n^T B\mathbf{x}_1}{\lambda_1-\lambda_n}\mathbf{x}_n\right\} \tag{1.2}$$

となりこちらは近接固有値の影響が色濃く出てくる.

1.3 模擬対角化法 (SD:Simulated Diagonalization)

固有値と固有ベクトルを求める問題を固有ベクトルに関する最適化問題に帰着させることを考える. 対称行列 $F(t)$ の異なる固有値 $\epsilon_\mu(t)$ に属する固有ベクトル $|\mu(t)>$ は規格直交化させることができる. 則ち $\mu=\nu$ で $<\mu(t)|\nu(t)>=1, \mu\neq\nu$ で $<\mu(t)|\nu(t)>=0$.
従って各固有ベクトルと固有値を求める問題は,

$$E[F(t), c_{i\mu}(t)] = \sum_{\mu\neq\nu}\left\{|<\mu(t)|F(t)|\nu(t)>|^2 + |<\mu(t)|\nu(t)>|^2\right\}$$

$$
\begin{aligned}
&+ \sum_{\mu} |<\mu(t)|\mu(t)> -1|^2 \\
&= \sum_{\mu\neq\nu} \left\{ \left| \sum_{ij} F_{ij}(t) c_{i\mu}(t) c_{j\nu}(t) \right|^2 + \left| \sum_{i} c_{i\mu}(t) c_{i\nu}(t) \right|^2 \right\} \\
&+ \sum_{\mu} \left| \sum_{i} c_{i\mu}(t)^2 - 1 \right|^2 \qquad (1.3)
\end{aligned}
$$

(右辺は $F(t)$ と $\{c_{i\mu}(t)\}$ に依存するのでこれらの依存性を陽にするために左辺を $E[F(t), c_{i\mu}(t)]$ と書いた) は一般の $\{c_{i\mu}(t)\}$ に対して $0 \leq E[F(t), c_{i\mu}(t)]$ で (常に正値か 0 を取るように右辺の各項の右肩に 2 乗がある), $\{c_{i\mu}(t)\}$ が $F(t)$ の真の固有ベクトル (=$|\mu(t)>$ の i 成分) となるとき

$$
0 = E[F(t), c_{i\mu}(t)] \qquad (1.4)
$$

を満たす. 固有値は $e_\mu(t) =<\mu(t)|F(t)|\mu(t)>$ で得られる.
さらに, 上から (下から)m 個の固有ベクトル・固有値だけを求める時には,

$$
\begin{aligned}
E[F(t), c_{i\mu}(t)] &= \sum_{\mu\neq\nu}^{m} \left\{ |<\mu|F(t)|\nu>|^2 + |<\mu|\nu>|^2 \right\} \\
&+ \sum_{\mu}^{m} \left\{ <\mu|F(t)|\mu> + |<\mu|\mu> -1|^2 \right\} \qquad (1.5)
\end{aligned}
$$

の様に全体を m 部分和にし, $\sum\{<\mu|F(t)|\mu>\}$ ($\approx$ 固有値の和) の項を付け加えることで実現できる.
以上で最適化問題に帰着された.

時間発展する系 $F(t)$ の固有ベクトル $\{c_{i\mu}(t)\}$ を一つ前の時間の系 $F(t-1)$ の真の固有ベクトル $\{c_{i\mu}(t-1)\}$ を使って高速に計算する問題への strategy を以下の様に考える:$F(t)$ の時間依存性は与えられていると仮定する.

$$
0 = E[F(t-1), c_{i\mu}(t-1)], \quad 0 < E[F(t), c_{i\mu}(t-1)] \qquad (1.6)
$$

である. 仮定により $F(t-1)$ と $F(t)$ はそれ程違わないために $t-1$ での固有ベクトル $\{c_{i\mu}(t-1)\}$ も t での固有ベクトル $\{c_{i\mu}(t)\}$ と余り違わない. この性質を使って $\{c_{i\mu}(t-1)\}$ から出発して $E[F(t1), c_{i\mu}(t)]$ (=0) の最小値に E や (下の) 式 (4) の右辺の計算値を使いながら逐次的に如何に速く漸近・到着する.
「如何に速く (= 高速化)」の問題は, 「(E や (下の) 式 (4) の右辺を 1 回計算する時間)×(逐次回数)= 固有値と固有ベクトルを得る時間」がどれ程速くなるかである.(下に述べるように)E や式 (4) の右辺は並列処理に適した形をしている. この 並列化で短くなった計算時間の効果を殺さないように少ない逐次回数で固有ベクトルを計算する のがミソである.

$E[F(t), c_{i\mu}(t')]$ の $\{c_{i\mu}(t')\}$ に関する最適化問題を解く方法は様々である. 例えば, 高次元化アルゴリズム [7] では,

$$
\begin{aligned}
\frac{d^2 c_{i\mu}(t')}{dt'^2} &= -\frac{\partial E}{\partial c_{i\mu}} \\
&= -4 \sum_{\nu(\neq\mu)} \left\{ \sum_{j} F_{ij} c_{j\nu} \left(\sum_{lm} F_{lm} c_{l\mu} c_{m\nu} \right) + c_{i\nu} \left(\sum_{l} c_{l\mu} c_{l\nu} \right) \right\} \\
&- 4 c_{i\mu} \left(\sum_{l} c_{l\mu}^2 - 1 \right) \qquad (1.7)
\end{aligned}
$$

を使う (t' は系の時間変化を与える t とは違うもの). また, 共役勾配法を使う方法などがある.

E や式 (1.5) の右辺は,(行列) × (行列),(行列) × (ベクトル) といった線形計算で高い並列性能とベクトル性能が期待される. 式 (4) の右辺第 1 項目 $\{\cdots\}$ で囲まれた初項からの寄与は一見すると 4 重の DO-Loop であるが $\sum_j F_{ij}c_{j\nu}$ が 2 回現れているので結局,$\{\cdots\}$ で囲まれた第 2 項からの寄与の部分と同様に 3 重の DO-Loop に帰着される. 右辺第 2 項は 1 重 DO-Loop である. 従って殆どの計算時間は式 (4) の右辺第 1 項の $\{\cdots\}$ の 3 重の DO-Loop からなる部分で費やされる (E も同様である). 行列の次数 N の関数としての計算量は, 重複する行列演算を考慮して $4N^3$ 回程度の乗算と加算を必要とし, 又, メモリサイズも $5N^2$ 個の実数型記憶領域を必要とし他の解法と比べるとやや多い.

1.4 並列性

ここでは実際に上記の方法で固有値と固有ベクトルが得られること, 式 (1.5) の右辺の 1 回の計算が並列化によりどれ程時間短縮されるかを報告する. 一様乱数を使った実対称密行列を使い, 行列サイズ (次数) は 500 から 4000 までのテスト行列を使った. スーパーコンピュータ SX-4/8(NEC 社) の測定環境で科学技術計算ライブラリとして ASL ライブラリ (NEC 社),IMSL ライブラリ, HOQRVW ルーチン [8] のソフトウェアを参考に比較を行なった. これら参考ソフトウェアの ASL は 1CPU でのみ動き (プログラムの詳細は公開されていない), 又, IMSL,HOQRVW も自動並列化には向かないアルゴリズムになっている. 一様乱数による対称行列 F での固有値と固有ベクトルを対象とした. 参照ソフトウェアとの消費 CPU 時間の比較を行った.1CPU では ASL のライブラリが良い性能を示しているが,SD アルゴリズムでは全体の 99% を占める行列演算に並列化とベクトル化を実行させることにより, 下記のような性能が得られた. [1]

行列の次数	500	1000	2000	4000
ASL(1CPU)	0.84 秒	5.4 秒	38 秒	285 秒
IMSL(1CPU)	2.4 秒	12 秒	67 秒	456 秒
HOQRVW(1CPU)	0.83 秒	6.2 秒	47 秒	358 秒
SD(1CPU)	0.67 秒	5.3 秒	42 秒	337 秒
SD(8CPU)	0.12 秒	0.79 秒	6.2 秒	49 秒

ASL は 1CPU で, どのアルゴリズムに比べ高速である. これは IMSL などに比べベクトル機で高速になるようにチューニングされているためと思われる. スーパーコンピュータに適した固有値ルーチンとして雑誌で紹介されていた HOQRVW は ASL とほぼ同程度の性能が得られている.SD は 1CPU では ASL, HOQRVW と同程度であるが,8CPU を使って計算を行なうと並列化が可能であるためにほぼ CPU 台数に比例した高速化が実現されている. また, 行列の次数に依らず高速化が達成されている. 例えば, 行列の次数 1000 ~ 4000 で 6.7-6.9 倍である. これらは 8CPU での計算であることを考えると理想的な並列化率に近い値である. 従って,SD では固有ベクトルを求めるのに逐次回数を高々 6.7-6.9 回以内で計算出来れば他のアルゴリズムに比べ高速に計算したことになる (固有値と固有ベクトルの計算の近似程度に依る).

1.5 考察

上記手法により最適化問題に帰着された固有値問題は, 任意の精度での計算を可能にし, また, 単純な線型計算による高い並列性により, 容易くベクトル・並列計算機の性能を引き出すことができた. 評価関数としてより演算量の少ない絶対値関数を取ることができる.

[1] ASL,ISML,HOQRVW は固有値と固有ベクトルを実際に計算しているのに対して, 此所での SD は式 (1.5) の右辺だけを 1 回だけ計算していることに注意せよ.

参考 & 文献:

(1) 森 正武, 名取 亮, 鳥居達生: 岩波講座 情報科学 18 数値計算, 岩波書店, 1981(3 章).

(2) 村田健郎, 小国 力, 唐木幸比古: スーパーコンピュータ - 科学技術計算への適用 -, 丸善,1985.

(3) 名取 亮: 数値解析とその応用, コロナ社,1990.

(4) 関口智嗣, 小柳義夫: 科学技術計算における並列化技術, 情報処理学会, Vol.27,No.9,1986.

(5) 片桐孝洋, 金田康正: 分散メモリ型並列計算機による Householder 法の性能評価, 情報処理学会,HPC 62-19,1996.

(6) 清水大志, 佐々木誠, 市原 潔, 岸田則生, 鈴木惣一朗, 佐藤 滋, 田中靖久, 横川三津夫, 他: 並列数値計算ライブラリの開発, 情報処理学会,HPC 62-22,1996.

(7) 新上和正, 下川信祐, 山田順一: 高次元化によるシステムの制御法, 電子情報通信学会総合大会 A-1-14,1997.

(8) 別府良孝: スーパーコンピュータに適した固有値ルーチン, bit 臨時増刊 (名取, 野寺 編), 共立出版,1987.

第 2 章

時間変動する系への適用

2.1 適用分野

本手法の適用分野について述べる.
将来的には

(I) (超) 並列機に適した行列の固有値と固有ベクトルの並列処理アルゴリズム, 則ち, 個々の CPU で計算処理の回数を多くし, 逆に CPU 間のデータ通信を少なくするアルゴリズムの開発

(II) 計算の ASIC 化 – 行列計算専用エンジン – の開発など

が考えられる. 前者ではソフトウェアレベルで後者はハードウェアレベルでの処理である. 特に (I) は科学技術計算が並列計算機を用いて行なわれる方向にあり, そのような計算機を最大限有効に利用するために重要と思われる.

さらに, 上記アルゴリズムは, 系の時間発展 (ダイナミクス) を調べるのに使われる. 例えば, 固体表面の粒子の拡散運動, ナノスケールの微細加工の動的過程, 半導体の結晶成長過程, 一般的には量子力学的系のダイナミクス, 量子化学での反応過程, また, ダイナミクスを利用して種々 (薬, 物質,$\cdots$) の設計など (主として電子状態と関連する) 広範囲な対象に適用される.

2.2 測定環境

次数 N で与えられる対称行列 $F(t)$ の全固有値と固有ベクトル $c_{i\mu}(t)$ を求める問題は, $E[F(t), c_{i\mu}(t)]$ の最小値 $(= 0)$ として得られる.[1](1) 「如何に速く (= 高速化)」の問題は, 「固有値と固有ベクトルを得る時間 $=(a) \times (b)$, 但し (a):E を 1 回計算する時間;(b): 逐次回数」で (a) の時間短縮と (b) の回数短縮がカギとなる. (a) の時間短縮の問題は並列化によりどれ程時間短縮されるかについて前報で得られた.(1) 行列サイズ (次数) は 500 から 4000 までのテスト行列でスーパーコンピュータ SX-4/8(NEC 社) の測定環境で CPU 数の比例した (8CPU で 6.7-6.9 倍の) 高速化が実現された. 従って,(b) の逐次回数の問題では SD では固有ベクトルを高々 6.7-6.9 回数以内で計算出来れば他のアルゴリズムに比べ高速に計算できたことになる. 以下で $E[F(t), c_{i\mu}(t')]$ の $\{c_{i\mu}(t')\}$ に関する最適化問題を解く方法として, 高次元化アルゴリズム (2)
$d^2 c_{i\mu}(t')/dt'^2 = -\partial E/\partial c_{i\mu}$ (t' は系の時間変化の t と違う) と, 章を改めて Newton 法系を使って問題に答えて行こう.

---

[1]小さい (大きい) 順から m $(\leq N)$ 個の固有値のみを求める場合には式 (1) を少し変形することで得られる:(1) で N を m と置き, 小さい順からの固有値を求める場合 (大きい順からの固有値を求める場合) に $\sum_{\mu=1}^{m} < \mu(t)|F(t)|\mu(t) >$ を左辺に加える (差し引く).

2.3 計算機シミュレーション

(I) $F(t)$ の時間発展: 行列の次数を N=500 とする. 先ず始めに SX-4 の計算ライブラリ ASL を使って行列の固有値と固有ベクトルを解き初期値とする. その後は SD(高次元化アルゴリズムとニュートン法で) と比較の為に ASL で同時に解く. SD では毎回前回に求まった固有ベクトルを新しい初期値として使用する. $F(t)$ の時間発展を $F_{ij}(t+\Delta) = F_{ij}(t)(1+\sin(2\pi\Delta)),(\Delta = 0.0025)$ と取る. 各時間ステップで 1.5% 程度行列 $F(t)$ の成分が大きくなる. また, 初期の $F(t_0)$ の各成分はサイコロを振って得られランダムな値になっている. これは, 時間的に変化する行列 $F(t)$ を (b) の逐次回数を 1 にして固有値と固有ベクトルがどの程度の近似で計算出来るかを問題にしていることに相当する.

(II) 収束判定と結果: 収束判定は,ASL と SD で計算した固有値と固有ベクトルに対して各々二つの差の 2 乗和の平方根を使って相対誤差 ($r_\epsilon(t)$: 固有値;$r_{vec}(t)$: 固有ベクトル) で判定する.$r_\epsilon(t)$ は時間を t_0 から始めて $t = t_0 + 64\Delta$ まではゆっくり増大する ($r_\epsilon(t) \leq 10^{-3}$) がその時間を境に再び減少する. 一方 $r_{vec}(t)$ は t_0 から始めて $t = t_0 + 64\Delta$ を越えてもゆっくりと増大し続ける. 時間ステップ Δ に対して増加分はほぼ $2 \cdot 10^{-4}$ 程度ある.(これは高次元化アルゴリズムの結果である. ニュートン法は収束に更に時間が掛かる.)

2.4 考察

SD では固有値と固有ベクトルで収束状況が異なる. 固有値はある範囲で近似的に収束しているように思われるが, 固有ベクトルは時間経るに従ってゆっくりであるが発散する. この為に長期に渡っては定期的に ASL で厳密な計算を行ないその結果を使って初期値を更新する必要がある. 従って, 仮に 64 回に一回 ASL を実行するならば SD は他のアルゴリズムに比べ依然として 6-7 倍高速であると結論される.

参考 & 文献:

(1) 新上和正, 下川信祐, 山田順一:'高次元化によるシステムの制御法', 電子情報通信学会総合大会 A-1-14,1997.

第 3 章

準 Newton 法の適用

3.1 非線形最適化問題

前章までの議論により, 対称行列の固有値問題が, 多変数の最適化問題に帰着された. 一般に最適化問題を解くには, 反復解法により

1. 適当な初期点 x_0 を与え, 繰り返しパラメータ k を 0 で初期化する
2. x_k の最適性の判定を行う
3. 新しい点 x_{k+1} の生成を行い, 繰り返しパラメータに 1 を加えて 2. に戻る

といった手続きを取るが, 「3. 新しい点の生成を行う」ための具体的な手続きとしては,

$$x_{k+1} = x_k + \alpha_k p_k \qquad (k = 0, 1, 2, \cdots) \tag{3.1}$$

といった計算が繰り返される. この時,p_k が探索方向で,α_k が探索幅である. まず, 探索方向の決定が問題になるが, 評価関数に関する多くの情報を用いる程, 1 回辺りの計算量は多くなるが, 収束までの反復回数が少なくて済む. 大きく別けて評価関数の勾配ベクトルまで用いる方法 (最急降下法) と評価関数の Hesse 行列まで用いる方法 (Newton 法) がある. これらは対照的な性質を持っており, 最急降下法では大域収束性が保証されている代りにその収束速度は遅く, 一方 Newton 法では解の近傍での収束速度は速いが (2 次収束), 初期点を解の近傍に取らなければ収束性は保証されない. また, 最急降下法はアルゴリズムが簡単であるのに対し,Newton 法では Hesse 行列の逆行列を計算しなければならないことにより,1 回の繰り返し計算あたりの計算量が多いという欠点がある. この両者の欠点を改良する方法として共役方向法と準 Newton 法が提案されている. 共役方向法は, 探索方向として共役な方向を探すことにより, 繰り返し計算の最終段階で 2 次元部分空間内の探索に陥ることを回避し, ある条件の元では大域的に収束し, 収束の次数は超 1 次であることがわかっている. 一方の準 Newton 法は勾配ベクトルを用いて Hesse 行列の逆行列に収束するような行列の列を構成し, 演算量を減らしながら大域的収束性と超 1 次収束を保証している.(1)

3.2 BFGS 法

ここでの問題は多変数の制約なし最適化問題なので, 準 Newton 法の中でも高速といわれている BFGS 法を用いた.(2)(3) 直線探索は, 大域収束性を満たすよう Wolfe の条件[1]を課す.

BFGS 法による収束の度合を示したものが、表 3.1である.

[1]探索幅を決める時の条件の一つで, ある程度以上の評価関数値の減少を求めた条件:$\nabla f(x^{(k+1)}) \geq (1 - \epsilon) f(x^{(k)})$

反復回数	SD	BFGS
0	-0.5404416922468461280OD+00	-0.5404416922468461280OD+00
50	-0.6254177235864512020OD+00	-0.1850908253282760100OD+01
100	-0.9767904524049120240OD+00	-0.1888318480224149940OD+01
150	-0.1382999142122948610OD+01	-0.1893884739168261610OD+01
200	-0.1822160888940331570OD+01	-0.1893872312654766520OD+01
250	-0.1773368332147589710OD+01	-0.1893871966607096890OD+01
300	-0.1873704426404889210OD+01	-
350	-0.1885548956390638020OD+01	-
400	-0.1885802677573303440OD+01	-
450	-0.1891557003381986180OD+01	-
500	-0.1890432976273254570OD+01	-
Ans.	-0.1893871966607097780OD+01	-0.1893871966607097780OD+01

表 3.1: 最小固有値の収束状況

よく知られているように、 BFGS 法は大域収束性と超一次収束性を持つ. Local Minimum に突っ込んで行く危険性をはらんでいるが、瞬く間にコスト関数値を下げているのがよくわかる.

また、収束条件を厳しくすると小数点以下 15 桁位の精度で一致する. これだけ精度が良いと近似解法というよりは、厳密解法である.

だたし、準 Newton 法特有の線形探索を行う際に、何度もコスト関数値の評価を行うので、反復計算 1 回あたりのコストは SD 法よりは高い. しかし、その分反復回数が少なくて済むので、全体的なコストは先の SD 法より小さなコストで精度の高い解を得ることができるようになった.

しかし、問題も残る. 反復計算時の更新ベクトルを求めるのに Hessian を用いているため、行列の次数の 4 乗に比例したメモリサイズを必要とする. これでは、十分に大きなシミュレーションができない. その点、 SD 法は、行列の次数の 2 乗に比例したサイズのメモリしか必要としない.

また、今回の報告では、並列処理をまったく考慮していない.

3.3 結論

今回用いた BFGS 法は、最小点を探索する際の更新ベクトルを算出するためにコスト関数の Hessian を用いている. そのため、小さな次元でさえ、大量のメモリを必要とするので、より大規模な計算を行うためには、 Limited memory BFGS 法の適用を検討しなければならない.

高速化に関しても、コスト関数値の計算や gradient の計算は高い並列性を持っている [(1)] ので、並列化による高速化が期待できる.

参考 & 文献:

(1) 名取 亮, 数値解析とその応用, コロナ社 (1992).

(2) 茨木 俊秀, 福島 雅夫, 岩波コンピュータサイエンス 最適化プログラミング (第 6 章),1991.

(3)D.G.Luenberger,Linear and Nonlinear Programming,2nd Edition,Addison-Wesley,,1989.

第 4 章

シミュレーションプログラム

4.1 プログラム構造

SD 法を高次元化アルゴリズムで解いた時のプログラムを添付する.

```
*--------------------------*
   PROGRAM STRUCTURE LIST
*--------------------------*
    (  ****:PROG.ENTRY  ++++:SUB.ENTRY  ####:TASK.ENTRY /MAIN.ENTRY/  (EXT.PROC
)  "MACRO.SUBR"  )

       ****:EIGEN


STRUCTURE NO:1
LINE     ****:EIGEN                                  : メインルーチン
           I
 000401    I--INITIAL1:<--D003                       : 初期化(時間ステップ１)
           I  I
 000194    I  I--RAND                                : 乱数発生
           I  I
 000240    I  I--RAND
           I  I
 000286    I  I--FORCE:<--D002                       : 力の計算
           I     I
 000051    I     I--FORCE1                           : 力の計算(部分)
           I     I
 000052    I     I--FORCE2                           : 力の計算(部分)
           I     I
 000069    I     I--FORCE3                           : 力の計算(部分)
           I
 000404    I--INITIAL2                               : 初期化(時間ステップ２以降)
           I
 000410    I--CPUTIME:<--D001                        : CPU 時間測定
           I  I
 000032    I  I--?ETIME?                             : 日づけ取得
           I
 000411    I--BCSMAA                                 : ライブラリのためのフィルター
           I  I
 000011    I  I--?DCSMAA?                            : 固有値計算ライブラリー
           I
 000412    I--CPUTIME:-->D001
           I
 000417    I--HOKAN                                  : 運動方程式を解く
           I  I
 000155    I  I--FORCE:-->D002                       : 力の計算(上述)
           I
 000421    I--CPUTIME:-->D001
           I
 000431    I--INITIAL1:-->D003
           I
 000433    I--INITIAL2
           I
 000435    I--CPUTIME:-->D001
```

4.2 ソースリスト

```
c234567
      subroutine BCSMAA
      include 'eigen.h'
c     local
c
      do j=1, n1
       do i=1, n1
        aslvec(i,j)=fock(i,j)
       enddo
      enddo
      call dcsmaa(aslvec,n1,n1,aslval(1),w1,err)
c
      write(6,100) aslval
 100  format(/'   eigenvalue  by dcsmaa:'/(5(' ',d27.20)))
c     write(6,200) aslvec
 200  format(/'   eigenvector by dcsmaa:'/(5(' ',f12.5)))
c
      return
      end
c
      subroutine cputime(sub)
C      real function etime(single)
C      single ... real single(2).
C                 single(1) = user time (sec)
C                 single(2) = system time (sec)
C      etime  ... user + system time (sec)
c
      character*6 sub
      dimension single(2)
      data time0 /0.0e+00/

      time1=etime(single)
      step=time1-time0
      icpu=nint(time1)

      ihr = icpu/3600
      imin= icpu/60-ihr*60
      isec= icpu-imin*60-ihr*3600

c     write(6,100) sub,step,ihr,imin,isec
      time0=time1
 100        format(/'< ',a6,' > ',' CPU time = ',f8.2,' Sec.',
     #           ' Total =',i4.2,'hr ',i2.2,'min ',i2.2,'sec.')

      return
      end
c234567
      subroutine force
      include 'eigen.h'
c
        call force1
        call force2
c
        do i=1,n1
        eig0(i)=f2(i,i)
        enddo
        do 30 i=1,n1
        f4(i)=-1.0d0+f3(i,i)
 30     continue
        do i=1,n1
        f2(i,i)=0.0d0
        f3(i,i)=0.0d0
        enddo
        do 40 j=1,n1
        do 40 i=1,n1
        ff3(i,j)=c1(i,j)*f4(j)
```



```
 40     continue
c
       call force3
c
       f1max=0.05d0
       f2max=0.05d0
       f3max=0.02d0
       do 60 j=1,n1
       do 60 i=1,n1
       if(f1max.lt.abs(ff1(i,j)))    f1max=abs(ff1(i,j))
       if(f2max.lt.abs(ff2(i,j)))    f2max=abs(ff2(i,j))
       if(f3max.lt.abs(ff3(i,j)))    f3max=abs(ff3(i,j))
 60    enddo
       v2=600.0d0/f2max
       v3=600.0d0/f3max
       v1=50.0d0/f1max
       return
       end
c234567
      subroutine force1
      include 'eigen.h'
c
      do j=1,n1
       do i=1,n1
        f1(i,j)=0.0d0
       enddo
      enddo
c
      do j=1,n1
       do k=1,n1
        do i=1,n1
         f1(i,j)=f1(i,j)+fock(i,k)*c1(k,j)
        enddo
       enddo
      enddo
c
      return
      end
c
      subroutine force2
      include 'eigen.h'
c
      do 20 j=1,n1
       do 20 i=1,n1
        f2(i,j)=0.0d0
        f3(i,j)=0.0d0
        do 20 k=1,n1
         f2(i,j)=f2(i,j)+c1(k,i)*f1(k,j)
         f3(i,j)=f3(i,j)+c1(k,i)*c1(k,j)
 20   continue
c
      return
      end
c
      subroutine force3
      include 'eigen.h'
c
      do 5 j=1,n1
       do 5 i=1,n1
        ff1(i,j)=0.0d0
        ff2(i,j)=0.0d0
 5    enddo
      do 50 j=1,n1
       do 50 k=1,n1
        do 50 i=1,n1
         ff1(i,j)=ff1(i,j)+f1(i,k)*f2(k,j)
```

```
          ff2(i,j)=ff2(i,j)+c1(i,k)*f3(k,j)
 50    continue
c
       return
       end
c234567
       subroutine hokan(bb,newfun)
       include 'eigen.h'
       real*8 newfun
c
       vv=0.0d0
       do 10 j=1,n1
        do 10 i=1,n1
         dxx(i,j)=c1(i,j)-c0(i,j)
      &          -dt2*(ff1(i,j)*v1+ff2(i,j)*v2+ff3(i,j)*v3)
         c2(i,j)=dxx(i,j)+c1(i,j)
         vx(i,j)=(c2(i,j)-c0(i,j))/dtt
         c2(i,j)=dtt*vx(i,j)*bb+c0(i,j)
         c0(i,j)=c1(i,j)
         c1(i,j)=c2(i,j)
         vv=vv+vx(i,j)*vx(i,j)
 10    enddo
       call force
       p1=0.0d0
       p2=0.0d0
       p3=0.0d0
       do 30 j=1,n1
        p3=p3+f4(j)*f4(j)
 30    enddo
       do 40 j=1,n1
        do 40 i=j+1,n1
         p1=p1+f2(i,j)*f2(i,j)
         p2=p2+f3(i,j)*f3(i,j)
 40    enddo
       newfun=(p1+p1)*2.0d0+p3
       t1=vv/(2.0d0*xn2)
       p1=p1/(xn1-1.0d0)
       p2=p2/(xn1-1.0d0)
       p3=p3/xn1
       tenrg=t1+p1+p2+p3
       return
       end
       subroutine initial1
       include 'eigen.h'
       dimension pj(n1+10),v(n1+10)
c
       if(icon.eq.100) goto 2000
       icon=100
       dt=0.0025d0
       dtt=2.0d0*dt
       dt2=dt*dt
       n2=n1*n1
       xn2=float(n2)
       xn1=float(n1)
       t1=1.0d0
       ii=16984
       bbs=0.995d0
c****** initial c1(i,j) ******************
       do 1 j=1,n1
        vv=0.0d0
        do 2 i=1,n1
         call rand(ii,xx)
         c1(i,j)=(xx-0.5d0)/0.5d0
         vv=vv+c1(i,j)*c1(i,j)
 2      enddo
        vv=1.0d0/sqrt(vv)
```



```
        do 3 i=1,n1
         c1(i,j)=c1(i,j)*vv
 3      enddo
 1     enddo
c****** Schmidt's orthogonarization ******************
       do 10 i=1,n1
        do 20 j=1,i-1
         pj(j)=0.0d0
 20     continue
        do 30 k=1,n1
         v(k)=c1(k,i)
 30     continue
        do 40 j=1,i-1
         do 50 k=1,n1
          pj(j)=pj(j)+v(k)*c1(k,j)
 50      continue
 40     continue
        do 70 j=1,i-1
         do 60 k=1,n1
          v(k)=v(k)-pj(j)*c1(k,j)
 60      continue
 70     continue
        ras=0.0d0
        do 80 k=1,n1
         ras=ras+v(k)*v(k)
 80     continue
        ras=sqrt(ras)
        ras=1.0d0/ras
        do 90 k=1,n1
         c1(k,i)=ras*v(k)
 90     continue
 10    continue
       do j=1,n1
        do i=1,n1
         vx(i,j)=1.0d0
         c0(i,j)=c1(i,j)-vx(i,j)*dt
        enddo
       enddo
c******* insert fock matrix ***************
       do j=1,n1
        do i=j,n1
         call rand(ii,xx)
         fock0(i,j)=(xx-0.5d0)*1.0d0
         df=alpha*sin(omega*dble(iter)+i+j)
         fock(i,j)=fock0(i,j)*(1.0d0+df)
         fock(j,i)=fock(i,j)
        enddo
       enddo
c ********************************************************* c
       write( 7,700) n1,imax,itime
  700  format(1h ,'7:Eigenvalues and Eigenstates (n,max,time=',i4,
      & ',',i4,',',i4,')')
       write( 7,*) '6 ',20*imax,' 7'
       write( 7,*) '1 2 0 dotcolor=1'
       write( 7,*) '1 3 0 dotcolor=2'
       write( 7,*) '1 4 0 dotcolor=3'
       write( 7,*) '1 5 0 dotcolor=4'
       write( 7,*) '1 6 0 dotcolor=5'
       write( 7,*) '1 7 0 dotcolor=6'
       write( 7,'(a,f6.4,a)') 'time (dt=',dt,')'
       write( 7,*) 't1'
       write( 7,*) 'eigenvalues'
       write( 7,*) 'fock energy'
       write( 7,*) 'off-diagonal energ.'
       write( 7,*) 'diagonal energ.'
       write( 7,*) 'total energ.'
```

```
c ********************************************************* c
      write( 8,800) nl,imax,itime
  800 format(1h ,'8:Eigenvalues and Eigenstates (n,max,time=',i4,
     & ',',i4,',',i4,')')
      write( 8,*) '2 ',20*imax,' 3'
      write( 8,*) '1 2 0 dotcolor=1'
      write( 8,*) '1 3 0 dotcolor=2'
      write( 8,'(a,f6.4,a)') 'time (dt=',dt,')'
      write( 8,*) 'cost function'
      write( 8,*) 'cost function by log10'
c ********************************************************* c
      write(10,1000) nl,imax,itime
 1000 format(1h ,'10:CPU Time & Eigenvalues by ASL & SD (n,max,time='
     & ,i4,',',i4,',',i4,')')
      write(10,*) '2 ',nl*itime,' 3'
      write(10,*) '1 2 0 dotcolor=1'
      write(10,*) '1 3 0 dotcolor=2'
      write(10,*) 'eigenstates'
      write(10,*) 'Eigenvalue (SD)'
      write(10,*) 'Eigenvalue (Library)'
c ********************************************************* c
      call force
      return
 2000 write( 7,*) '@@@@@'
      write( 8,*) '@@@@@'
      write(10,*) '@@@@@'
      do i=1,nl
      xlow=100000.0d0
      do j=i,nl
      bb=eig0(j)
      if(bb.le.xlow) k=j
      if(bb.le.xlow) xlow=bb
      enddo
      xx=eig0(i)
      eig0(i)=xlow
      eig0(k)=xx
      enddo
      return
      end
c234567
      subroutine initial2
      include 'eigen.h'
c
      if(icon.eq.100) goto 2000
       icon=100
       dt=0.0025d0
       dtt=2.0d0*dt
       dt2=dt*dt
       n2=nl*nl
       xn2=float(n2)
       xnl=float(nl)
       t1=0.1d0
       ii=26984
       bbs=1.005d0
       do j=1,nl
        do i=1,nl
         vx(i,j)=0.1d0
         c0(i,j)=c1(i,j)-vx(i,j)*dt
        enddo
       enddo
c******* insert fock matrix ****************
       do j=1,nl
        do i=j,nl
         df=alpha*sin(omega*dble(iter)+i+j)
         fock(i,j)=fock0(i,j)*(1.0d0+df)
         fock(j,i)=fock(i,j)
```



```
        enddo
       enddo
c ********************************* c
       if(iter.eq.2)then
        write(8,1000) nl,imax,itime
 1000 format(1h ,'8:Eigenvalues and Eigenstates (n,max,time=',i4,
     & ',',i4,',',i4,')')
        write(8,*) '5 ',imax*20*(itime-1),' 6'
        write(8,*) '1 2'
        write(8,*) '1 3'
        write(8,*) '1 4'
        write(8,*) '1 5'
        write(8,*) '1 6'
        write(8,*) 'time (dt=0.0025)'
        write(8,*) 't1'
        write(8,*) 'eigenvalues'
        write(8,*) 'fock energy'
        write(8,*) 'off-diagonal energ.'
        write(8,*) 'diagonal energ.'
       endif
       return
c ********************************* c
 2000 write(6,*) '@@@@@'
      do i=1,nl
       xlow=100000.0d0
       do j=i,nl
        bb=eig0(j)
        if(bb.le.xlow) k=j
        if(bb.le.xlow) xlow=bb
       enddo
       xx=eig0(i)
       eig0(i)=xlow
       eig0(k)=xx
       do l=1,nl
        xx=c1(l,i)
        yy=c0(l,i)
        c1(l,i)=c1(l,k)
        c0(l,i)=c0(l,k)
        c1(l,k)=xx
        c0(l,k)=yy
       enddo
      enddo
      return
      end
c234567
      program eigen
      include 'eigen.h'
      real*8 newfun
c ********************** c
c *** Initialization *** c
c ********************** c
      iter2=0
      do i=-10,nl+10
       eig0(i)=0.0d0
       aslval(i)=0.0d0
      enddo
c *** matrix parameter *** c
      alpha=0.37d0
      omega=0.0393d0
c *** time step *** c
      itime=1
c *** iteration max *** c
      imax1=4000
      imax2=2
c ************************************* c
      do 5000 iter=1,itime
```

```
      tt00=0.00001d0
      icon=0
      if(iter.eq.1) then
       imax=imax1
       call initial1
      else
       imax=imax2
       call initial2
      endif
      istep=imax1/400
      if(istep.eq.0) istep=1
      in1=n1/20
      if(in1.eq.0) in1=1
      call cputime('initial')
      call BCSMAA
      call cputime('ASL    ')
      do 6000 i=1,imax
       iter2=iter2+1
       bb=bbs
       if(t1.lt.tt00) bb=sqrt(tt00/t1)
       call hokan(bb,newfun)
c --- debug
       write(8,800) i,newfun,log10(newfun)
       ii=i-istep*((i-1)/istep)
       call cputime('run hokan')
       if(ii.ne.1) goto 6000
       do k=1,n1,in1
        write(7,700) i,t1,eig0(k),p1,p2,p3,tenrg
       enddo
 6000  continue
      do i=1,n1
       write(10,1000) i,eig0(i),aslval(i)
      enddo
      if(iter.eq.1) then
       call initial1
      else
       call initial2
      endif
      call cputime('s1     ')
 5000 continue
      write(6,100) eig0
  100 format(/'   eigenvalue  by dcsmaa:'/(5(' ',d27.20)))
c     write(6,200) aslvec
  200 format(/'   eigenvector by dcsmaa:'/(5(' ',f12.5)))
c ***************************** c
  700 format(1h , i8,6f20.10)
  800 format(1h ,i8,2f30.20)
  900 format(1h ,i8,5f20.10)
 1000 format(1h ,i8,2f20.10)
      stop
      end
c234567
      subroutine rand(ii,xx)
      implicit real*8 (a-h,o-z)
      l10=18972
      l11=451
      l12=89766
      ii=l10+l11*ii
      ii=ii-l12*((ii-1)/l12)
      xx=float(ii)/float(l12)
      return
      end
```