ISSN 1349-1121
JAXA-RM-13-018

宇宙航空研究開発機構研究開発資料
JAXA Research and Development Memorandum

固有直交分解と統計的推定法を応用した高サンプリングレート流速プローブと低サンプリングレートPIV計測データを用いた高サンプリングレート流れ場オフライン推定C++コードの開発

青木 良尚

2014年3月

宇宙航空研究開発機構
Japan Aerospace Exploration Agency

固有直交分解と統計的推定法を応用した 高サンプリングレート流速プローブと低サンプリングレートPIV計測データを用いた 高サンプリングレート流れ場オフライン推定C++コードの開発*

青木　良尚*1

Development of the High Sampling Rate Flow Field Offline Estimation C++ Code that Applies Proper Orthogonal Decomposition and Stochastic Estimation using High Sampling Rate Velocity Probe Data and Low Sampling Rate PIV Data*

Yoshihisa AOKI*1

Abstract

Three step method is the offline estimation method of the flow field using control theory. C++ code of the three-step method which doesn't depend on any operating systems and applications was developed so I open this code to public. Three-step method is offline estimation method for time-resolved velocity fields using time-resolved point measurements and non-time-resolved particle image velocimetry data. This code was validated with the data in Florida State University and I confirmed the result agreed with the result from MATLAB code.

Key Words: Proper Orthogonal Decomposition, System Identification, Fluid Dynamics, Kalman Smoother

概要

スリーステップ法と呼ばれる，制御理論を応用した流れ場のオフライン推定法がある．オペレーティングシステムやアプリケーションに依存せずに，スリーステップ法による流れ場のオフライン推定が可能となるように，スリーステップ法のC++コードを開発したので，これを公開する．スリーステップ法は，固有直交分解と統計的推定法を応用して，高サンプリングレート流速プローブと低サンプリングレートPIV計測データから，高サンプリングレート流れ場の離散線形状態方程式を同定し，このシステムに固定区間カルマンスムーザを適用して高サンプリングレート流れ場をオフライン推定する方法である．このコードは，フロリダ州立大学が所有するデータを用いて検証され，MATLABで作成されたコードと同じ推定結果が得られることを確認した．

１．はじめに [1)]

流体制御は，流れの剥離や境界層遷移などの流体現象を意図的にコントロールしようとする技術である．航空機の高速化と航続距離の延長につながる巡航抵抗低減を目的とした境界層乱流遷移遅延制御の研究の歴史は古く，１９３０年代からNACAにおいて受動制御（自然層流翼に代表される）と能動制御（境界層吸い込み等を用いた層流境界層制御に代表される）の両面から研究が行われた．エンジン性能の急速な進歩と，境界層乱流遷移遅延制御の実用化への技術的な困難さからこの研究は下火となったが，

*　平成25年12月20日受付（Received 20 December, 2013）
*1　航空本部　風洞技術開発センター（Wind Tunnel Technology Center, Institute of Aeronautical Technology）

オイルショックによる燃料費高騰をきっかけに，巡航抵抗低減が注目され再び研究が活発化した．自然層流翼は，工作精度や保守に課題があった為，第二次世界大戦期の軍用機を除いて量産機に採用されなかったが，本田技研工業が量産機への採用に適する自然層流翼を開発し，ビジネスジェット機に採用した[2]．現在，この機体は FAA の型式証明取得を目指した試験が行われており，実用化が近い．剥離制御としての能動的境界層制御は，日本においても US-2 に採用され，既に実用化されている．一方で，能動的な境界層乱流遷移遅延制御は，エネルギー収支，アクチュエータの複雑な構造や重く複雑な制御システムなど課題が多く，実用化されていない．こうした中，1990 年代後半には，能動的な境界層乱流遷移遅延制御よりも性能が劣るが課題を改善可能な，自然層流翼と能動制御を組み合わせるハイブリッド制御が NASA によって提案された．ハイブリッド制御は，現在，最も実用化に近い能動的境界層制御である．

これまで，能動的な流体制御は流体の安定性理論の発達とともに発達していった．これに加えて，近年のコンピュータの発達に伴い複雑な制御が可能になったことで，より高い制御効果を得るため，能動的な流体制御に対して制御理論の導入が進められている．流体制御への制御理論の応用を目的とした研究の中で，固有直交分解を応用した非定常流れ場推定法であるスリーステップ法[3]の，MATLAB 等のアプリケーションに依存しない C++コードの開発を行ったので，これを公開する．スリーステップ法は，二次元カルマン渦列の実験から計測された PIV による低サンプリングレートの流れ場データと，ある位置に設置されたプローブによる高サンプリングレートの非定常流速データを用いて，高サンプリングレートの流れ場を表現する離散線形状態方程式モデルを同定し，このシステムに対してカルマンスムーザを適用して高サンプリングレートの流れ場情報を推定する方法である．

２．スリーステップ法とその周辺の数学的準備

２．１　固有直交分解（Proper Orthogonal Decomposition, POD）の基礎[4]

POD は，データのコンパクトな表現を得るための，多変数の統計的手法である．この方法は，高次元データの低次元化や，特徴抽出に用いられる．POD の考え方は，多数の相関のある変数を，元々の値に対する誤差を出来るだけ小さくしながら，より少数の相関の無い変数に減らすことである．これは，データの共分散行列の固有値を基礎とする直交変換によって，データをこの固有値に対応する固有ベクトルの張る部分空間に展開する．この変換は，変数間の相関を無くすとともに，分散を最大化する．

POD の最も大きな特徴は，元々のデータと低次元化された線形表現の二乗平均誤差を最小化することである．POD は，非線形問題にも適用されることがあるが，POD から得られるのは，データを表現する仮想空間において最適に近似される線形多様体である．POD が線形であることは，線形作用素が利用できるという利点があるが，データが非線形多様体に属する場合，これは主要な制限にもなる．

POD の数学的な表現について述べる．領域Ωに属するデータを$\theta(x,t)$とする．このデータは，平均値$\mu(x)$と時間変動成分$\vartheta(x,t)$に分解できる．

$$\theta(x,t)=\mu(x)+\vartheta(x,t) \qquad (1)$$

ある時間t_kにおける瞬時場を，スナップショット$\theta^k(x)=\vartheta(x,t_k)$とする．POD は，データ$\theta(x,t)$のスナップショット全体の特徴的な構造$\varphi(x)$を得ることを目指す．これは，下式を満たす基底関数を見つ

けることと等価である.

$$Maximize \quad \left\langle \left| \left(\vartheta^k, \varphi \right) \right|^2 \right\rangle \; with \; \left\| \varphi \right\|^2 = 1 \qquad (2)$$

$\left(f, g \right) = \int_\Omega f(x) g(x) d\Omega$ ：領域 Ω における内積

$\left\langle \cdot \right\rangle$ ：平均作用素

$\left\| \cdot \right\| = \left(\cdot, \cdot \right)^{\frac{1}{2}}$ ：ノルム

$\left| \cdot \right|$ ：絶対値

式（２）は，変動成分 ϑ を基底 φ に沿って展開する場合，他の基底関数に沿って展開するよりも大きい平均エネルギーを含むことを意味する．制約条件 $\left\| \varphi \right\|^2 = 1$ は，解を一意とするために課される．この制約条件下における式（２）の解は，ラグランジュの未定乗数法で計算できる.

$$J[\varphi] = \left\langle \left| (\vartheta, \varphi) \right|^2 \right\rangle - \lambda \left(\left\| \varphi \right\|^2 - 1 \right) \qquad (3)$$

式（３）は，微分値が０の時に最大値を取るので，式（２）の条件は，次の積分固有値問題と等価となる.

$$\int_\Omega \left\langle \vartheta^k(x) \, \vartheta^k(x') \right\rangle \varphi(x') dx' = \lambda \varphi(x) \qquad (4)$$

$\left\langle \vartheta^k(x) \, \vartheta^k(x') \right\rangle$ ：平均自己相関関数

式（２）の最適化問題の解は，POD モードや固有直交モード（Proper Orthogonal Modes, POMs）と呼ばれる,式(４)の積分方程式の直交固有関数 $\varphi_i(x)$ によって与えられる. 対応する固有値 $\lambda_i \left(\lambda_i \geq 0 \right)$ を，固有直交値（Proper Orthogonal Values, POVs）と呼ぶ．POM は，変動成分 $\vartheta(x,t)$ を分解するための基底として用いられる.

$$\vartheta(x,t) = \sum_{i=1}^{\infty} a_i(t) \varphi_i(x) \qquad (5)$$

$$a(t) = \left(\vartheta(x,t), \varphi_i(x) \right)$$

$$\left\langle a_i(t)a_j(t)\right\rangle = \delta_{ij}\,\lambda_i$$

最も大きなPOVに対するPODモードは,スナップショット全体を特徴付ける最適なベクトルとなる.データに含まれるエネルギーεは，POV の和として定義される（$\varepsilon = \sum_j \lambda_j$）．また，$i$番目の POD モードによって捉えられるエネルギーの割合は，$\lambda_i / \sum_j \lambda_j$ によって与えられる．

２．２　POD スナップショット法

POD スナップショット法は，離散化された POD である．ある時間t_mにおけるM個のスナップショットの集合を考える．

$$u^{(m)}(x) = u(x,t_m), \quad m = 1,2,\ldots,M \qquad (6)$$

平均値を下式とする．

$$u_0(x) = \frac{1}{M}\sum_{m=1}^{M} u^{(m)}(x) \qquad (7)$$

データは，平均値と変動成分に分解できる．

$$u(x,t_m) = u_0(x) + u'(x,t_m) \qquad (8)$$

変動成分の自己相関の離散形式より，POM$\phi_i(x)$と POVλ_iに関する式（４）に対応する下式を得られる．

$$C\,\phi_i(x) = \lambda_i\,\phi_i(x) \qquad (9)$$

$C = \frac{1}{M}\left(u'(x,t_m)u'(x,t_n)\right)$：離散自己相関行列

この直交基底を用いて，変動成分は下式に分解される．

$$u'(x,t_m) = \sum_{i=1}^{M} a_i(t_m)\phi_i(x) \qquad (10)$$

$$a_i(t_m) = \left(u'(x,t_m)\phi_i(x)\right)$$

$$\left\langle a_i(t_m) a_j(t_m) \right\rangle = \delta_{ij}\,\lambda_i$$

次に，重み係数を導入したスナップショット法について述べる．
データ行列 X を下式とする．

$$X = \left[u'(x,t_0) \quad u'(x,t_1) \quad \cdots \quad u'(x,t_m)\right] \qquad (11)$$

データ行列 X の内積の重み行列を W とすると，自己相関行列 C は下式となる．

$$C = X^T W X \qquad (12)$$

ここで，例えば二次元流れ場データに対して，重み行列 W を各グリッドの面積 $diag\left(dx_1 dy_1, \cdots, dx_n dy_n\right)$ とすると，W のベクトルノルムは，運動エネルギーの積分量と解釈できる．

$$\left\| u'(x,t_j) \right\|_2^2 = \left(u'(x,t_j)\right)^T W\, u'(x,t_j) = \iint \left(u'(t_j)^2 + v'(t_j)^2\right) dx\, dy \qquad (13)$$

この時，POM $\phi_j(x)$ と POD 係数 $a_i(t_j)$ は，式（１２）の自己相関行列の特異値分解を $C = V \Sigma V^T$（V ：直交行列，Σ ：対角行列）とすると，下式となる．

$$\Phi = \left[\phi_1(x) \quad \cdots \quad \phi_m(x)\right] = X V \Sigma^{-\frac{1}{2}} \qquad (14)$$

$$a = \begin{bmatrix} a_1(t_1) & & a_1(t_m) \\ \vdots & \cdots & \vdots \\ a_n(t_1) & & a_n(t_m) \end{bmatrix} = \Sigma^{\frac{1}{2}} V^T$$

$$\phi_j^T W \phi_i = \delta_{ij}$$

２．３　POD と統計的推定法による低次元化推定（Linear Stochastic Estimation POD, LSE-POD）

まず初めに一般的な統計的推定法について述べる．統計的推定値を，統計学の知識を用いた条件付き平均の近似値の期待値とする．事象 P の下で発生する事象 S の条件付き期待値は，下式の確率 p と等価である．

$$\langle S \mid P = p \rangle = \int s \frac{f_{SP}(s,p)}{f_P(p)} ds \qquad (15)$$

$\langle \cdot \rangle$：期待値

f_{SP}：事象 S と事象 P の結合確率密度関数

f_P：事象 P 単独の確率密度関数

式（１５）の近似解法として，テイラー級数展開を用いた条件付き平均の統計的推定法が提案されている．

$$\hat{s}_i = \langle S_i \mid P_j = p_j \rangle = A_{ij} p_j + B_{ijk} p_j p_k + \cdots \qquad (16)$$

$\hat{s}_i$：条件付き平均の推定値

線形統計的推定（Linear Stochastic Estimation, LSE）では，式（１６）の線形項のみを使用する．スリーステップ法では，非定常圧力センサの計測値 $p_j(t)$ の条件下での流れ場 $u_i(t)$ を推定する．この時，流れ場の LSE による推定値は，下式となる．

$$\hat{u}_i(t) = A_{ij} p_j(t) \qquad (17)$$

式（１７）の係数 A_{ij} は，流れ場 $u_i(t)$ と流れ場の推定値 $\hat{u}_i(t)$ の二乗平均誤差を最小化するように，最小二乗法を用いて求めることができる．

式（１７）は，複数の非定常圧力センサを想定しているが，１つの非定常圧力センサの時系列計測値を用いる事ができる．これを，時間遅れ線形統計的推定（delay time-LSE）と呼ぶ．

$$\hat{u}_i(t) = A_{ij} p(t - \tau_j) \qquad (18)$$

典型的な PIV による流れ場データは，多大な計測点数を持つ．ここで，全ての流れ場 $u_i(t)$ の代わりに，POD を用いて低次元化した流れ場の POD 係数 $a_i(t)$ の線形統計的推定値を LSE によって求める．これを，低次元化推定（Linear Stochastic Estimation-POD, LSE-POD，又は，modified-LSE, mLSE）と呼ぶ．

$$\hat{a}_i(t) = A_{ij} p_j(t) \qquad (19)$$

これもまた，１つの非定常圧力センサの時系列計測値を用いることが出来る．これを，時間遅れ低次元

化推定（delay time-LSE-POD, delay time-mLSE）と呼ぶ.

$$\hat{a}_i(t) = A_{ij}\, p(t-\tau_j) \qquad (20)$$

２．４　システムの入力が無い離散線形システムのカルマンスムーザ [5)]

スリーステップ法では，システムの入力が無い場合の離散カルマンスムーザを用いる．まず，離散システムを下式とする.

$$x_k = F\, x_{k-1} + d_k \qquad (21)$$
$$\eta_k = H_k x_k + n_k$$

x_k ：離散システムの状態変数

η_k ：離散システムの観測値

d_k ：プロセスノイズ

n_k ：センサノイズ

H_k ：観測行列

ここで，プロセスノイズとセンサノイズは白色ノイズ（$E\left[d_i d_j^T\right] = Q_i \delta_{i-j}$，$E\left[n_i n_j^T\right] = R_i \delta_{i-j}$，$E\left[d_i n_j^T\right] = 0$）とする.

まず，カルマンフィルタによって推定を行う．初期値を下式とする.

$$\hat{x}_{f,0}^{+} = E\left[x_0\right] \qquad (22)$$
$$P_{f,0}^{+} = E\left[\left(x_0 - \hat{x}_0^{+}\right)\left(x_0 - \hat{x}_0^{+}\right)^T\right]$$

ここで，P は推定誤差の共分散である．次に，式（２２）を初期値として，下式を用いて状態推定を行う.

$$P_{f,k}^{-} = F\, P_{f,k-1}^{+} F^T + Q_{k-1} \qquad (23)$$
$$K_{f,k} = P_{f,k}^{-} H_k^T \left(H_k P_{f,k}^{-} H_k^T + R_k\right)^{-1}$$
$$\hat{x}_{f,k}^{-} = F\, \hat{x}_{f,k-1}^{+}$$
$$\hat{x}_{f,k}^{+} = \hat{x}_{f,k}^{-} + K_{f,k}\left(\eta_k - H_k \hat{x}_{f,k}^{-}\right)$$

$$P_{f,k}^{+} = \left(I - K_{f,k}H_k\right)P_{f,k}^{-}$$

次に，カルマンフィルタによる推定値を用いて，固定区間のカルマンスムーザによる推定を行う．まず，初期値を下式とする．

$$\hat{x}_{s,Nt} = \hat{x}_{f,Nt}^{+} \qquad (24)$$

$$P_{s,Nt} = P_{f,Nt}^{+}$$

式（２４）を初期値として，下式を用いて状態推定を行う．

$$L_{f,k+1}^{-} = \left(P_{f,k+1}^{-}\right)^{-1} \qquad (25)$$

$$K_{s,k} = P_{f,k}^{+}F^{T}L_{f,k+1}^{-}$$

$$P_{s,k} = P_{f,k}^{+} - K_{s,k}\left(P_{f,k+1}^{-} - P_{s,k+1}\right)K_{s,k}^{T}$$

$$\hat{x}_{s,k} = \hat{x}_{f,k}^{+} + K_{s,k}\left(\hat{x}_{s,k+1} - \hat{x}_{f,k+1}^{-}\right)$$

以上より，入力が無い離散システムにおける，カルマンスムーザによる状態推定値が求まる．

２．５　スリーステップ法

スリーステップ法は，低サンプリングレートの PIV 計測により得られた流れ場データと，ある位置に設置されたプローブによる高サンプリングレートの非定常定常流速データから，高サンプリングレートの流れ場を推定する方法である．スリーステップ法は，以下の 3 段階の手順からなる．

1) 非定常流れ場の POD 係数の統計的推定

delay time mLSE を用いて，低サンプリングレートの流れ場データと高サンプリングレートのある位置の非定常流速データから，高サンプリングレートの非定常流れ場 POD 係数の統計的推定値を求める．

2) 非定常流れ場の離散線形システムの同定

LSE を用いて，非定常流れ場 POD 係数の統計的推定値から，流れ場の構造を考慮した，流れ場のダイナミクスを表現する離散線形システムを同定する．

3) カルマンスムーザによる POD 係数の推定

流れ場の離散線形システムと，低サンプリングレートの流れ場データに対してカルマンスムーザを適用し，高サンプリングレートの非定常流れ場の推定値を求める．

1 段階目の非定常流れ場の初期推定の後に，カルマンスムーザによる推定を行うのは，初期推定値に含まれるセンサノイズの影響を低減するためである．

３．スリーステップ法によるカルマン渦列の非定常流れ場推定

スリーステップ法を用いて，図 1 の後流の非定常流れ場推定を行う．ここで，一様流流速は 4.2m/s，低サンプリングレートの PIV データのサンプリングレートは 32Hz，計測点数は 2317 点，高サンプリングレートの流速データのサンプリングレートは 800Hz である．

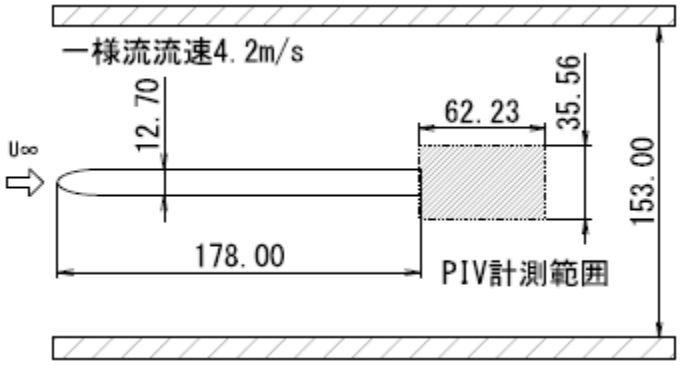


図 1　風洞試験概要図

まず最初のステップとして，低サンプリングレートの PIV による流れ場データと，ある位置に設置されたプローブによる高サンプリングレートの非定常流速データを用いて，非定常流れ場の POD 係数の統計的推定を行う．統計的推定値を求める準備として，POD を用いて PIV による流れ場データの低次元化を行う．ここでは，POD の次数を 7 次とした．この場合，図 2 の通り得られた POD 係数 $a_i(n)(i=1,\ldots,7)$ によって推定される流れ場に含まれる運動エネルギーは全体の 89.4%であるが，カルマン渦列を再現するためには十分である．

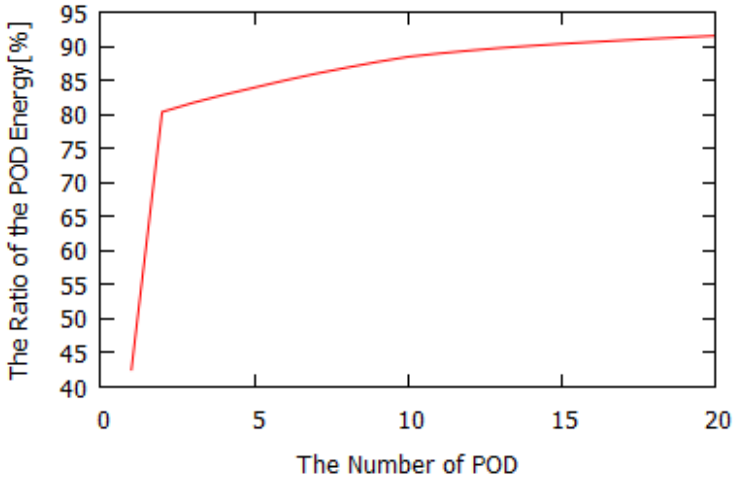


図 2　使用する POD モード数に対する含まれるエネルギーの割合

この低サンプリングレートの流れ場の POD 係数と高サンプリングレートのある点の非定常流速データ $v(n)$ から，delay time-mLSE を用いて，高サンプリングレートの非定常 POD 係数の統計的推定値 $\hat{a}_i(n)$ を求める．

$$\hat{a}_i(n) = \sum_{j=-\tau}^{\tau} A_{ij} v(n+j) \qquad (25)$$

τ ：時間遅れ

式（２５）の時間遅れは，図３のように推定された POD 係数と計測された POD 係数から計算可能な，流れ場の運動エネルギーの推定誤差を最小化する時間遅れが存在するので，この最適値を用いる．ここでは，時間遅れ τ を 5 とした．

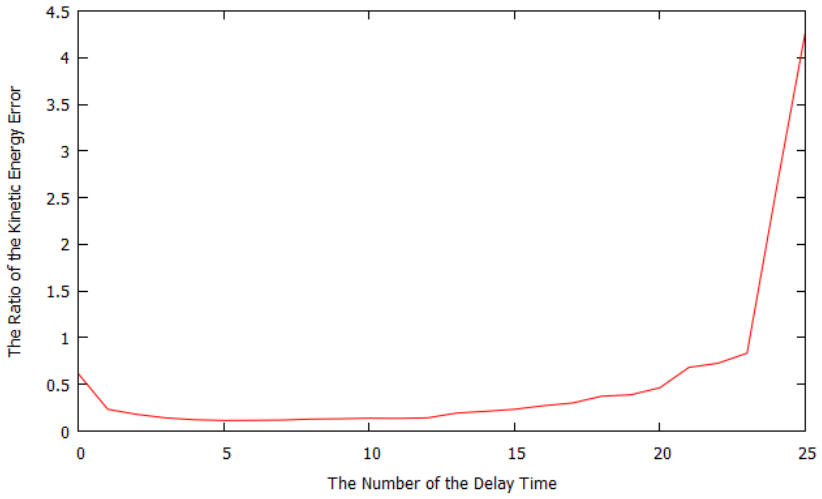


図３　delay time-mLSE の時間遅れに対する流れ場のエネルギー誤差

次のステップとして，式（２５）から得られた非定常流れ場 POD 係数の統計的推定値から，流れ場の構造を考慮した，高サンプリングレートの流れ場のダイナミクスを表現する離散線形システムを同定する．

カルマン渦列の場合，事前知識として，1 番目と 2 番目の POD 係数で表現される周期的振動モードと，その他のモードから表現される事がわかっているので，LSE を用いて，下式のように，振動モードの部分システムと，その他のモードの部分システムを分けて，それぞれ同定する．

$$\begin{bmatrix} \hat{a}_{OSC}(n) \\ \hat{a}_{LSE}(n) \end{bmatrix} = \begin{bmatrix} F^{OSC} & 0 \\ 0 & F^{LSE} \end{bmatrix} \begin{bmatrix} \hat{a}_{OSC}(n-1) \\ \hat{a}_{LSE}(n-1) \end{bmatrix} \qquad (26)$$

$$\hat{a}_{OSC}(n) = \begin{bmatrix} \hat{a}_1(n) \\ \hat{a}_2(n) \end{bmatrix}$$

$$\hat{a}_{LSE}(n)=\begin{bmatrix}\hat{a}_3(n)\\ \hat{a}_4(n)\\ \hat{a}_5(n)\\ \hat{a}_6(n)\end{bmatrix}$$

次に，振動モードの部分システム F^{OSC} が不安定にならないように， F^{OSC} の固有値の絶対値が０．９９９となるように F^{OSC} を定数倍する．以上により，カルマン渦列の高サンプリングレート流れ場の離散線形システムが同定できた.

最後のステップとして，１段階目で得られた低サンプリングレートの流れ場データと高サンプリングレートの POD 係数の統計的推定値を観測値として，式（２６）の離散線形システムに式（２３），（２４）のカルマンスムーザを適用し，高サンプリングレートの非定常流れ場の推定値を求める．観測値の共分散の初期値 $P^{+}_{f,0}$ とノイズの共分散（Q_k， R_k）を下式とした.

$$P^{+}_{f,0}=5I \qquad (27)$$

$$Q_k=diag(1,1,0.004,0.5,0.5,0.5)$$

$$R_k=\begin{cases}2\times10^{4}\,I\\ 1\times10^{-10}\,I\end{cases}$$

I ：サイズ 5×5 の単位行列

以上により，低サンプリングレートの流れ場データと，高サンプリングレートのある点の流速から，高サンプリングレートの流れ場を推定できた.

非定常流速データのノイズを下式とした時，POD の統計的推定値とカルマンスムーザによる推定値における流れ場の運動エネルギー誤差の割合は，図４となる．全てのノイズの値でカルマンスムーザによる推定値の誤差は統計的推定値の誤差を下回る．従って，カルマンスムーザによる推定はノイズに対してロバストであることがわかる.

$$\gamma=\left(\frac{n'_{RMS}}{v'_{RMS}}\right)^2 \qquad (28)$$

n'_{RMS} ：非定常流速データのノイズの RMS

v'_{RMS} ：非定常流速データの変動成分の RMS

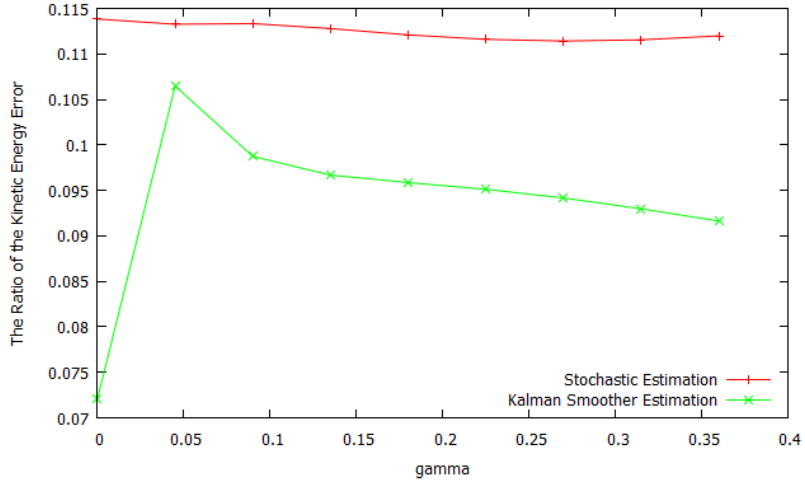


図４　高サンプリングレート流速プローブのノイズに対する推定 POD 係数のエネルギー誤差

謝辞

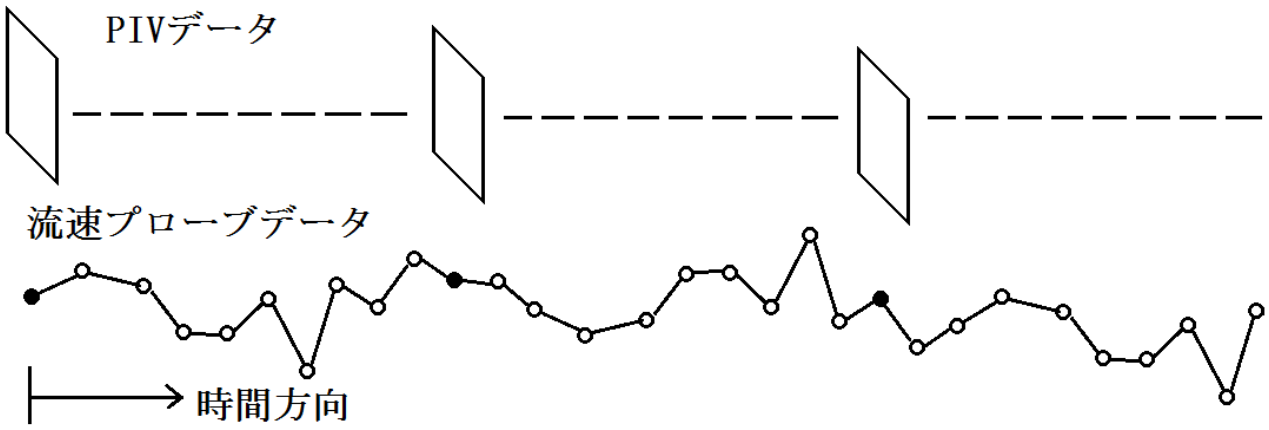
この開発は，２０１２年１１月１日から２０１３年１０月３１日の宇宙航空研究開発機構の長期在外研修の一部として，アメリカ合衆国フロリダ州立大学 Florida Center for Advanced Aero-Propulsion（FCAAP）で実施した．フロリダ州立大学教授及び FCAAP Associate Director である，Dr. Louis Cattafesta には，この在外研修において受入ホストを務めて頂いた．研修を進めるに当たり，航空本部事業推進部の久野克子氏，竹本勇介氏には事務手続きについてご助力を頂いた．航空本部風洞技術開発センター計画管理室の内藤朋子氏には，事務手続きの代理を務めて頂いた．航空本部風洞技術開発センターの渡辺重也センター長（現，航空本部チーフエンジニア），浜本滋計画管理チーフマネージャ（現，センター長），満尾和徳超音速風洞/フラッタ風洞セクションリーダ（現，研究計画マネージャ事務代理）には，研修計画についてご助言を頂いた．また，超音速風洞/フラッタ風洞セクション及び宇宙輸送ミッション本部宇宙輸送系システム技術研究開発センターの調布在勤の方々には，不在の間の仕事を引き継いで頂いた．末筆ながら，ここに感謝の意を表する．

参考文献

1) Ronald D. Joslin, “Overview of Laminar Flow Control”, NASA/TP-1998-208705
2) Fujino. M, “Design and Development of the HondaJet”, AIAA-2003-2530
3) Jonathan H. Tu, Clarence W. Rowley, John Griffin, Louis Cattafesta, Adam Hart, Lawrence S. Ukeiley, “Integration of non-time-resolved PIV and time-resolved velocity point sensors for dynamic estimation of time-resolved velocity fields”, AIAA-2012-033
4) GAETAN KERSCHEN, JEAN-CLAUDE GOLINVAL, ALEXANDER F. VAKAKIS and LAWRENCE A, BERGMAN, “The Method of Proper Orthogonal Decomposition for Dynamical Characterization and Order Reduction of Mechanical Systems：An Overview, Nonlinear Dynamics”, 2005, 41, 147-169
5) 片山　徹, 「新版応用カルマンフィルタ」, 朝倉書店, 2000

付録.　スリーステップ法によるカルマン渦列のカルマンスムーザによるオフライン推定 C++コード

スリーステップ法による，高サンプリングレート非定常流速プローブと，低サンプリングレート PIV データを用いた，高サンプリングレートのカルマン渦列の流れ場を推定する C++コードを記載する．C++コードは，”main.cpp”，”matclass.cpp”，”matclass.hpp”の３つのファイルから成る．入力データは，第１引数に高サンプリングレート非定常流速プローブデータファイル名（文字列），第２引数に非定常流速プローブのサンプリングレート(Hz，整数)，第３引数に低サンプリングレート PIV データファイル名（文字列），第４引数に PIV データのサンプリングレート(Hz, 整数)，第５引数にスナップショットに含まれる PIV データ数(vx+vy, 整数)，第６引数に計測時間（秒で指定，整数），第７引数に delay time-mLSE の時間遅れ（整数）とする．それぞれのデータは，下図のように同期して計測する必要が有る．



付図１　流速プローブデータ及び PIV データの同期計測

データフォーマットはテキスト形式である．流速プローブデータは，１列の行列とする．また，PIV データは，行方向を位置，列方向を時間として，カンマ区切りのテキスト形式とする．但し，流速プローブ及び各位置における PIV データの平均値がゼロとなるように，それぞれのデータから平均値を引いておくこと．コードのデータ読み込み部分を修正すれば，どんなデータフォーマットにも対応可能である．推定された流れ場は，ファイル名”result.csv”のファイルに，PIV と同一のフォーマットで保存される．推定された流れ場のサンプリングレートは，流速プローブのサンプリングレートと同一となる．その他，一般的な注意点としては，POD スナップショットの重み係数を１に，使用する POD モード数を 7 に固定しているほか，コードの開発では実行速度やメモリ使用量，エラー処理については深くは考慮していない．

例えば，GNU Compiler Collection の g++で最も単純にプログラムを作成するためには，下記のコマンドを打てばよい．

g++ main.cpp matclass.cpp matclass.hpp

OS が Windows の場合，実行可能形式ファイル名が”a.exe”，流速プローブデータファイル名”probe.csv”，流速プローブのサンプリングレート 800Hz, PIV データファイル名”piv.csv”, PIV サンプリングレート 32Hz, データ数 4634，計測時間１秒，delay time-mLSE の時間遅れ τ を 5，データファイルが exe ファイルと同一のディレクトリにある場合，プログラムを実行するためには下記のコマンドを打てばよい．

a probe.csv 800 piv.csv 32 4634 1 5

```
// “main.cpp”
// This program is free software: you can redistribute it and/or modify
// it under the terms of the GNU General Public License as published by
// the Free Software Foundation, either version 3 of the License, or
// (at your option) any later version.
//
// This program is distributed in the hope that it will be useful,
// but WITHOUT ANY WARRANTY; without even the implied warranty of
// MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
// GNU General Public License for more details.
//
// You should have received a copy of the GNU General Public License
// along with this program.  If not, see <http://www.gnu.org/licenses/>.

#include <iostream>
#include <cstdio>
#include <complex>
#include <cstdlib>

#include "matclass.hpp"

//SNUM is the maximum number of the characters
#define SNUM 20000

int main(int argc, char* argv[])
{
	int i,j,k,m,n,sp,num,pnum,delay,rep,kk,sec,fpp,fpiv,dnum,dn;
	double dx,dy,tempd,*vec1,*vec2,*pdata;
	complex<double> *eigen_val;
	matclass *drms,*amat,*bmat,*cmat,*fit_podc,*fit_coef,*dmodel,*kf_podc,*kfm_podc;
	matclass **pm,**pp,*kg,*qmat,*rmat0,*rmat1,*umat,*ks_podc,*ps;
	FILE *fp;
	char str[SNUM],*tok;

	printf("Probe data file name:%s¥n",argv[1]);
	printf("Probe sampling rate:%s[Hz]¥n",argv[2]);
	printf("PIV data file name:%s¥n",argv[3]);
	printf("PIV sampling rate:%s[Hz]¥n",argv[4]);
	printf("The Number of the data in one PIV snapshot:%s¥n",argv[5]);
	printf("Measurement time:%s[sec]¥n",argv[6]);
	printf("Delay time:%s¥n",argv[7]);

	dx = dy = 1.;
	delay = atoi(argv[7]);//delay time
	pnum = 7;// the number of POD
	sec = atoi(argv[6]);

	//if this routine is modified, this code can be used to any import file format data.
```

```
//drms(i,j) = u_i(j)
puts("Import Data");
fpp = atoi(argv[2]);
pdata = new double[sec*fpp];
fp = fopen(argv[1],"r");
if(fp == NULL)
{
	printf("Can't open %s¥n",argv[1]);
	return 1;
}
for(i=0;i<sec*fpp;i++)
{
	memset(str,0,SNUM);
	fgets(str,SNUM,fp);
	pdata[i] = atof(str);
}
fclose(fp);

dnum = atoi(argv[5]);
fpiv = atoi(argv[4]);
dn = fpp / fpiv;
num = sec*fpiv-1;
drms = new matclass(dnum,num+1);
fp = fopen(argv[3],"r");
if(fp == NULL)
{
	printf("Can't open %s¥n",argv[3]);
	return 1;
}
for(i=0;i<dnum;i++)
{
	memset(str,0,SNUM);
	fgets(str,SNUM,fp);
	tok = strtok(str,",¥n");
	(*drms)(i,0) = atof(tok);
	for(j=1;j<num+1;j++)
	{
		tok = strtok(NULL,",¥n");
		(*drms)(i,j) = atof(tok);
	}
}
fclose(fp);
drms->set_weight(dx*dy);
drms->pod(pnum);
//data import finishes.
//////////////////////

//dynamic model
```

```
puts("Construct Dynamic Model");
amat = new matclass(num,delay*2+1);
bmat = new matclass(num,pnum);
cmat = new matclass(delay*2+1,pnum);
for(j=0;j<num;j++)
{
        for(k=0;k<delay*2+1;k++)
                (*amat)(j,k) = pdata[(j+1)*dn-delay+k];
}
for(j=0;j<num;j++)
{
        for(k=0;k<pnum;k++)
                (*bmat)(j,k) = drms->podc(k,j+1);
}
tempd = amat->lsq(bmat,cmat);

cmat->transpose();

fit_coef = new matclass(*cmat);

delete amat;
delete bmat;
delete cmat;

fit_podc = new matclass(pnum,sec*fpp);
for(k=delay;k<sec*fpp-delay;k++)
{
        for(j=0;j<pnum;j++)
        {
                for(int kk=-delay;kk<=delay;kk++)
                        (*fit_podc)(j,k) += (*fit_coef)(j,kk+delay) * pdata[k+kk];
        }
}

amat = new matclass(sec*fpp-2*delay,pnum-2);
bmat = new matclass(sec*fpp-2*delay,pnum-2);
cmat = new matclass(pnum-2,pnum-2);
for(k=delay;k<sec*fpp-delay-1;k++)
{
        for(j=2;j<pnum;j++)
        {
                (*amat)(k-delay,j-2) = (*fit_podc)(j,k);
                (*bmat)(k-delay,j-2) = (*fit_podc)(j,k+1);
        }
}
amat->lsq(bmat,cmat);
cmat->transpose();
dmodel = new matclass(pnum,pnum);
```

```
for(i=2;i<pnum;i++)
{
	for(j=2;j<pnum;j++)
		(*dmodel)(i,j) = (*cmat)(i-2,j-2);
}
delete amat;
delete bmat;
delete cmat;
amat = new matclass(sec*fpp-2*delay,2);
bmat = new matclass(sec*fpp-2*delay,2);
cmat = new matclass(2,2);
for(k=delay;k<sec*fpp-delay-1;k++)
{
	for(j=0;j<2;j++)
	{
		(*amat)(k-delay,j) = (*fit_podc)(j,k);
		(*bmat)(k-delay,j) = (*fit_podc)(j,k+1);
	}
}
amat->lsq(bmat,cmat);
cmat->transpose();
eigen_val = new complex<double>[2];
tempd = ((*cmat)(0,0)-(*cmat)(1,1))*((*cmat)(0,0)-(*cmat)(1,1))+4.*(*cmat)(0,1)*(*cmat)(1,0);
if(tempd > 0.)
{
	eigen_val[0] = complex<double>(((*cmat)(0,0)+(*cmat)(1,1))/2.+sqrt(tempd)/2.,0.);
	eigen_val[1] = complex<double>(((*cmat)(0,0)+(*cmat)(1,1))/2.-sqrt(tempd)/2.,0.);
}
else
{
	eigen_val[0] = complex<double>(((*cmat)(0,0)+(*cmat)(1,1))/2.,sqrt(-tempd)/2.);
	eigen_val[1] = complex<double>(((*cmat)(0,0)+(*cmat)(1,1))/2.,-sqrt(-tempd)/2.);
}
tempd = abs(eigen_val[0]);
delete amat;
delete bmat;
for(i=0;i<2;i++)
{
	for(j=0;j<2;j++)
		(*dmodel)(i,j) = (*cmat)(i,j) / tempd * 0.999;
}

delete cmat;
delete[] eigen_val;

//kalman filter
puts("Kalman Filter Estimation");
kf_podc = new matclass(pnum,sec*fpp);
```

```
for(i=0;i<pnum;i++)
        (*kf_podc)(i,delay) = (*fit_podc)(i,delay);
kfm_podc = new matclass(pnum,sec*fpp);
pp = new matclass*[sec*fpp];
pm = new matclass*[sec*fpp];
for(i=0;i<sec*fpp;i++)
{
        pp[i] = new matclass(pnum,pnum);
        pm[i] = new matclass(pnum,pnum);
}
for(i=0;i<pnum;i++)
        (*pp[delay])(i,i) = 5.;
amat = new matclass(pnum,pnum);
bmat = new matclass(pnum,pnum);
cmat = new matclass(pnum,pnum);
qmat = new matclass(pnum,pnum);
(*qmat)(0,0) = 1.;
(*qmat)(1,1) = 1.;
(*qmat)(2,2) = 0.004;
(*qmat)(3,3) = 0.5;
(*qmat)(4,4) = 0.5;
(*qmat)(5,5) = 0.5;
(*qmat)(6,6) = 0.5;
rmat0 = new matclass(pnum,pnum);
rmat1 = new matclass(pnum,pnum);
for(i=0;i<pnum;i++)
{
        (*rmat0)(i,i) = 20000.;
        (*rmat1)(i,i) = 1e-10;
}
kg = new matclass(pnum,pnum);
umat = new matclass(pnum,pnum);
for(i=0;i<pnum;i++)
        (*umat)(i,i) = 1.;
for(i=delay+1;i<sec*fpp-delay;i++)
{
        amat->matrix_mult(*dmodel,*pp[i-1]);
        bmat->matrix_mult(*amat,*dmodel,0,1);
        pm[i]->matrix_add(*bmat,*qmat);

        if(i%dn == 0)
                amat->matrix_add(*pm[i],*rmat1);
        else
                amat->matrix_add(*pm[i],*rmat0);
        kg->matrix_mult_inv(*pm[i],*amat,0,1);

        for(j=0;j<pnum;j++)
        {
```

```
                for(k=0;k<pnum;k++)
                        (*kfm_podc)(j,i) += (*dmodel)(j,k) * (*kf_podc)(k,i-1);
        }

        if(i%dn == 0)
        {
                for(j=0;j<pnum;j++)
                {
                        (*kf_podc)(j,i) = (*kfm_podc)(j,i);
                        for(k=0;k<pnum;k++)
                                (*kf_podc)(j,i) += (*kg)(j,k)* (drms->podc(k,i/dn) - (*kfm_podc)(k,i));
                }
        }
        else
        {
                for(j=0;j<pnum;j++)
                {
                        (*kf_podc)(j,i) = (*kfm_podc)(j,i);
                        for(k=0;k<pnum;k++)
                                (*kf_podc)(j,i) += (*kg)(j,k) * ((*fit_podc)(k,i) - (*kfm_podc)(k,i));
                }
        }

        amat->matrix_sub(*umat,*kg);
        pp[i]->matrix_mult(*amat,*pm[i]);
}

//kalman smoother
puts("Kalman Smoother Estimation");
ks_podc = new matclass(pnum,sec*fpp);
for(i=0;i<pnum;i++)
        (*ks_podc)(i,sec*fpp-1) = (*kf_podc)(i,sec*fpp-1);
ps = new matclass(*pp[sec*fpp-delay-1]);
for(i=sec*fpp-delay-1;i>=delay;i--)
{
        amat->matrix_mult(*pp[i],*dmodel,0,1);
        kg->matrix_mult_inv(*amat,*pm[i+1],0,1);

        amat->matrix_sub(*pm[i+1],*ps);
        bmat->matrix_mult(*kg,*amat);
        cmat->matrix_mult(*bmat,*kg,0,1);
        ps->matrix_sub(*pp[i],*cmat);

        for(j=0;j<pnum;j++)
        {
                (*ks_podc)(j,i) = (*kf_podc)(j,i);
                for(k=0;k<pnum;k++)
                        (*ks_podc)(j,i) += (*kg)(j,k) * ((*ks_podc)(k,i+1) - (*kfm_podc)(k,i+1));
```

```
		}
	}

	delete amat;
	delete bmat;
	delete cmat;
	delete qmat;
	delete kg;
	delete rmat0;
	delete rmat1;
	delete umat;
	delete ps;

	//calcurate velocity from POD coefficients
	puts("Calcurate Estimated Velocity");
	amat = new matclass(dnum,sec*fpp);
	vec1 = new double[pnum];//POD coefficients
	vec2 = new double[dnum];//velocity
	for(j=delay;j<sec*fpp-delay;j++)
	{
		for(i=0;i<pnum;i++)
			vec1[i] = (*ks_podc)(i,j);
		drms->vel(vec1,vec2);
		for(i=0;i<dnum;i++)
			(*amat)(i,j) = vec2[i];
	}
	puts("Output Result");
	fp = fopen("result.csv","w");
	for(i=0;i<dnum;i++)
	{
		for(j=0;j<sec*fpp;j++)
			fprintf(fp,"%lf,",(*amat)(i,j));
		fprintf(fp,"¥n");
	}
	fclose(fp);

	delete amat;
	delete[] vec1;
	delete[] vec2;

	delete drms;
	delete fit_podc;
	delete fit_coef;
	delete dmodel;
	delete kf_podc;
	delete kfm_podc;
	for(i=0;i<sec*fpp;i++)
	{
```

```
                delete pp[i];
                delete pm[i];
        }
        delete[] pp;
        delete[] pm;
        delete ks_podc;
        delete pdata;

        return 0;
}


//matclass.hpp
#ifndef _MATCLASS_DEFINED__
#define _MATCLASS_DEFINED__

#include <iostream>
#include <cmath>
#include <cstring>

#define TOR 1e-6
#define PI 3.14159265358979

using namespace std;

class matclass
{
  public:
  //least square method:
  // Ax=b,A is this matrix, b is matclass b, x is matclass c, returned values are stored in matclass c
  double lsq(matclass *b, matclass *c);
  //return POD modes Phi_m,n
  double phi(int m, int n);
  //return POD coefficients r_m,n
  double podc(int m, int n);
  //calcurate velocity from POD coefficients
  void vel(double *podc, double *v);
  //data(matclass data) is convert to POD coefficients(matclass podc)
  int podc(matclass *data, matclass *podc);
  //matclass a - matclass b is stored in this matclass
  void matrix_sub(matclass a, matclass b);
  //matclass a + matclass b is stored in this matclass
  void matrix_add(matclass a, matclass b);
  //matclass a * matclass b is stored in this matclass
  //if matclass a is inversed, ia != 0
  //if matclass b is inversed, ib != 0
  void matrix_mult_inv(matclass a, matclass b, int ia, int ib);
  //matclass a * matclass b is stored in this matclass
```

```
  //if matclass a is transposed, ia != 0
  //if matclass b is transposed, ib != 0
  void matrix_mult(matclass a, matclass b);
  //matclass a * matclass b is stored in this matrix
  void matrix_mult(matclass a, matclass b, int ta, int tb);
  //return matrix size, m is row, n is column
  void getsize(int *m, int *n);
  //calculate singular value decomposition of this matclass and store here
  int svd();
  //calculate proper orthogonal decomposition of this matclass and store here
  //the number of the POD modes is npod
  int pod(int npod);
  //weight matrix for POD is stored as all the same value
  void set_weight(double val);
  double& operator()(int m, int n)
  {
        m_tempd = 0.;
        if(m > m_m || 0 > m)
              return m_tempd;
        if(n > m_n || 0 > n)
              return m_tempd;
        return m_mat[m][n];
  };
  //transpose this matclass
  void transpose();
  ~matclass();
  //allocated matrix size m*n
  matclass(int m, int n);
  //allocated matrix size as same as matclass mat
  //and initial values are the same values as matclass mat
  matclass(matclass& mat);

  protected:
  int svd(double tor, int snum, int flag);
  int pod(int npod, double tor);
  int pod_(int npod, double tor);
  void delete_all();
  matclass();
  double **m_mat,**m_tmat,*m_weight,**m_lmat,**m_rmat,*m_singular,*m_eigen,**m_emat,**m_phi,**m_podc;
  double m_tempd;
  int m_m,m_n,m_snum,m_enum,m_pnum;
};

#endif
```

//matclass.cpp

```
#include "matclass.hpp"

int matclass::podc(matclass *data, matclass *podc)
{
	int md,nd,mp,np,i,j,k;

	if(m_pnum <= 0)
		return 0;

	data->getsize(&md,&nd);
	podc->getsize(&mp,&np);

	if(md != m_m)
		return 0;
	if(nd != np)
		return 0;
	if(mp != m_pnum)
		return 0;

	for(i=0;i<m_pnum;i++)
	{
		for(j=0;j<nd;j++)
		{
			(*podc)(i,j) = 0.;
			for(k=0;k<m_m;k++)
				(*podc)(i,j) += m_phi[k][i] * m_weight[k] * (*data)(k,j);
		}
	}

	return 1;
}

void matclass::vel(double *podc, double *v)
{
	int i,j;
	double ret = 0.;

	for(i=0;i<m_m;i++)
	{
		v[i] = 0.;
		for(j=0;j<m_pnum;j++)
			v[i] += m_phi[i][j] * podc[j];
	}
}

int matclass::svd(double tor, int snum, int flag)
{
	int i,j,k,num;
```

```
double *vec1,*vec2,err,**rmat,**tmat,eval1,eval2,r;

if(m_snum > 0)
{
	for(i=0;i<m_m;i++)
		delete[] m_lmat[i];
	delete[] m_lmat;
	for(i=0;i<m_n;i++)
		delete[] m_rmat[i];
	delete[] m_rmat;
	delete[] m_singular;
}

m_lmat = new double*[m_m];
for(i=0;i<m_m;i++)
{
	m_lmat[i] = new double[m_m];
	memset(&m_lmat[i][0],0,m_m*8);
}
m_rmat = new double*[m_n];
for(i=0;i<m_n;i++)
{
	m_rmat[i] = new double[m_n];
	memset(&m_rmat[i][0],0,m_n*8);
}

if(m_m >= m_n)
{
	m_snum = m_n;

	if(snum > m_snum)
		snum = m_snum;
	else if(snum < 1)
	{
		m_snum = 0;
		return 0;
	}

	tmat = new double*[m_snum];
	for(i=0;i<m_snum;i++)
	{
		tmat[i] = new double[m_snum];
		memset(&tmat[i][0],0,m_snum*8);
	}
	vec1 = new double[m_snum];
	vec2 = new double[m_snum];
	if(flag == 0)
	{
```

```
		for(i=0;i<m_snum;i++)
		{
			for(j=0;j<m_snum;j++)
			{
				for(k=0;k<m_m;k++)
					tmat[i][j] += m_mat[k][i] * m_mat[k][j];
			}
		}
	}
	else
	{
		for(i=0;i<m_snum;i++)
		{
			for(j=0;j<m_snum;j++)
			{
				for(k=0;k<m_m;k++)
					tmat[i][j] += m_mat[k][i] * m_weight[k] * m_mat[k][j];
			}
		}
	}

	m_singular = new double[snum];
	memset(m_singular,0,snum*8);

	for(i=0;i<snum;i++)
	{
		memset(vec1,0,m_snum*8);
		vec1[i] = 1.;
		memset(vec2,0,m_snum*8);
		for(j=0;j<m_snum;j++)
		{
			for(k=0;k<m_snum;k++)
				vec2[j] += tmat[j][k] * vec1[k];
		}
		eval2 = 0.;
		for(j=0;j<m_snum;j++)
			eval2 += vec1[j] * vec2[j];
		r = 0.;
		for(j=0;j<m_snum;j++)
			r += vec2[j] * vec2[j];
		if(r > 0.)
		{
			r = sqrt(r);
			for(j=0;j<m_snum;j++)
				vec1[j] = vec2[j] / r;
		}
		eval1 = eval2;
```

```
num = 0;
err = tor + 1.;
while(err > tor && num < 300*m_m)
{
	memset(vec2,0,m_snum*8);
	for(j=0;j<m_snum;j++)
	{
		for(k=0;k<m_snum;k++)
			vec2[j] += tmat[j][k] * vec1[k];
	}

	eval2 = 0.;
	for(j=0;j<m_snum;j++)
		eval2 += vec1[j] * vec2[j];
	r = 0.;
	for(j=0;j<m_snum;j++)
		r += vec2[j] * vec2[j];
	if(r > 0.)
	{
		r = sqrt(r);
		for(j=0;j<m_snum;j++)
			vec1[j] = vec2[j] / r;
	}
	err = fabsl(eval1 - eval2);
	eval1 = eval2;

	num++;
}

for(j=0;j<m_snum;j++)
	m_rmat[i][j] = vec1[j];
memset(vec2,0,m_snum*8);
for(j=0;j<m_snum;j++)
{
	for(k=0;k<m_snum;k++)
		vec2[j] += tmat[j][k] * vec1[k];
}
eval2 = 0.;
for(j=0;j<m_snum;j++)
	eval2 += vec1[j] * vec2[j];
for(j=0;j<m_snum;j++)
{
	for(k=0;k<m_snum;k++)
		tmat[j][k] -= vec1[j] * vec1[k] * eval2;
}
m_singular[i] = sqrt(eval2);

}
```

```
		for(i=0;i<m_m;i++)
		{
			for(j=0;j<snum;j++)
			{
				m_lmat[i][j] = 0.;
				for(k=0;k<m_n;k++)
					m_lmat[i][j] += m_mat[i][k] * m_rmat[j][k];
			}
		}
		for(j=0;j<snum;j++)
		{
			if(fabsl(m_singular[j]) > tor)
			{
				for(i=0;i<m_m;i++)
					m_lmat[i][j] /= m_singular[j];
			}
		}
		for(j=m_n;j<m_m;j++)
		{
			for(i=0;i<m_m;i++)
				m_lmat[i][j] = 0.;
		}
		m_snum = snum;
}
else
{
		m_snum = m_m;
		if(snum > m_snum)
			snum =	m_snum;
		else if(snum < 1)
		{
			m_snum = 0;
			return 0;
		}
		tmat = new double*[m_snum];
		for(i=0;i<m_snum;i++)
		{
			tmat[i] = new double[m_snum];
			memset(&tmat[i][0],0,m_snum*8);
		}
		vec1 = new double[m_snum];
		vec2 = new double[m_snum];
		if(flag == 0)
		{
			for(i=0;i<m_snum;i++)
			{
				for(j=0;j<m_snum;j++)
				{
```

```
                                for(k=0;k<m_n;k++)
                                        tmat[i][j] += m_mat[i][k] * m_mat[j][k];
                        }
                }
        }
        else
        {
                for(i=0;i<m_snum;i++)
                {
                        for(j=0;j<m_snum;j++)
                        {
                                for(k=0;k<m_n;k++)
                                        tmat[i][j] += m_mat[i][k] * m_weight[k] * m_mat[j][k];
                        }
                }
        }
        m_singular = new double[snum];
        memset(m_singular,0,snum*8);

        for(i=0;i<snum;i++)
        {
                memset(vec1,0,m_snum*8);
                vec1[i] = 1.;
                memset(vec2,0,m_snum*8);
                for(j=0;j<m_snum;j++)
                {
                        for(k=0;k<m_snum;k++)
                                vec2[j] += vec1[k] * tmat[k][j];
                }
                eval2 = 0.;
                for(j=0;j<m_snum;j++)
                        eval2 += vec2[j] * vec1[j];
                r = 0.;
                for(j=0;j<m_snum;j++)
                        r += vec2[j] * vec2[j];
                if(r > 0.)
                {
                        r = sqrt(r);
                        for(j=0;j<m_snum;j++)
                                vec1[j] = vec2[j] / r;
                }
                eval1 = eval2;

                num = 0;
                err = tor + 1.;
                while(err > tor && num < 300*m_m)
                {
                        memset(vec2,0,m_snum*8);
```

```
                for(j=0;j<m_snum;j++)
                {
                        for(k=0;k<m_snum;k++)
                                vec2[j] += vec1[k] * tmat[k][j];
                }

                eval2 = 0.;
                for(j=0;j<m_snum;j++)
                        eval2 += vec2[j] * vec1[j];
                r = 0.;
                for(j=0;j<m_snum;j++)
                        r += vec2[j] * vec2[j];
                if(r > 0.)
                {
                        r = sqrt(r);
                        for(j=0;j<m_snum;j++)
                                vec1[j] = vec2[j] / r;
                }
                err = fabsl(eval1 - eval2);
                eval1 = eval2;

                num++;
        }

        for(j=0;j<m_snum;j++)
                m_lmat[j][i] = vec1[j];
        memset(vec2,0,m_snum*8);
        for(j=0;j<m_snum;j++)
        {
                for(k=0;k<m_snum;k++)
                        vec2[j] += vec1[k] * tmat[k][j];
        }
        eval2 = 0.;
        for(j=0;j<m_snum;j++)
                eval2 += vec1[j] * vec2[j];
        for(j=0;j<m_snum;j++)
        {
                for(k=0;k<m_snum;k++)
                        tmat[j][k] -= vec1[j] * vec1[k] * eval2;
        }
        m_singular[i] = sqrt(eval2);
}
for(i=0;i<snum;i++)
{
        for(j=0;j<m_n;j++)
        {
                m_rmat[i][j] = 0.;
                for(k=0;k<m_m;k++)
```

```
                                        m_rmat[i][j] +=  m_lmat[k][i] * m_mat[k][j];
                        }
                }
                for(i=0;i<snum;i++)
                {
                        if(fabsl(m_singular[i]) > tor)
                        {
                                for(j=0;j<m_n;j++)
                                        m_rmat[i][j] /= m_singular[i];
                        }
                }
                for(i=m_m;i<m_n;i++)
                {
                        for(j=0;j<m_n;j++)
                                m_rmat[i][j] = 0.;
                }
                m_snum = snum;
        }

        for(i=0;i<m_snum;i++)
                delete[] tmat[i];
        delete[] tmat;
        delete[] vec1;
        delete[] vec2;

        return num;
}

int matclass::svd()
{
        if(m_m >= m_n)
                return svd(TOR,m_n,0);
        else
                return svd(TOR,m_m,0);
}

int matclass::pod(int npod, double tor)
{
        int i,j,k,num;

        num = svd(tor,npod,1);

        if(m_pnum > 0)
        {
                for(i=0;i<m_m;i++)
                        delete[] m_phi[i];
                delete[] m_phi;
                for(i=0;i<m_pnum;i++)
```

```
                        delete[] m_podc[i];
                delete[] m_podc;
        }

        if(npod > m_n)
                m_pnum = m_n;
        else
                m_pnum = npod;
        m_phi = new double*[m_m];
        for(i=0;i<m_m;i++)
                m_phi[i] = new double[m_pnum];
        m_podc = new double*[m_pnum];
        for(i=0;i<m_pnum;i++)
                m_podc[i] = new double[m_n];
        for(i=0;i<m_pnum;i++)
        {
                for(j=0;j<m_n;j++)
                        m_podc[i][j] = m_singular[i] * m_rmat[i][j];
        }
        for(i=0;i<m_m;i++)
        {
                for(j=0;j<m_pnum;j++)
                {
                        m_phi[i][j] = 0.;
                        for(k=0;k<m_n;k++)
                                m_phi[i][j] += m_mat[i][k] * m_rmat[j][k];
                        if(fabsl(m_singular[j]) > tor)
                                m_phi[i][j] /= m_singular[j];
                        else
                                m_phi[i][j] = 0.;
                }
        }

        return num;
}

int matclass::pod(int npod)
{
        return pod(npod,TOR);
}

void matclass::set_weight(double val)
{
        int i;

        for(i=0;i<m_m;i++)
                m_weight[i] = val;
}
```

```
matclass::matclass()
{
	m_m = 0;
	m_n = 0;
	m_snum = 0;
	m_enum = 0;
	m_pnum = 0;
}

void matclass::transpose()
{
	int i,j,m,n;
	double **tmat;

	m = m_m;
	n = m_n;
	tmat = new double*[m_m];
	for(i=0;i<m_m;i++)
	{
		tmat[i] = new double[m_n];
		memcpy(&tmat[i][0],&m_mat[i][0],m_n*8);
	}

	delete_all();

	m_mat = new double*[n];
	for(i=0;i<n;i++)
		m_mat[i] = new double[m];
	m_m = n;
	m_n = m;

	for(i=0;i<n;i++)
	{
		for(j=0;j<m;j++)
			m_mat[i][j] = tmat[j][i];
	}

	for(i=0;i<m_m;i++)
		delete[] tmat;
	delete[] tmat;
}

matclass::matclass(int m, int n)
{
	int i;

	m_mat = new double*[m];
```

```
	for(i=0;i<m;i++)
	{
		m_mat[i] = new double[n];
		memset(&m_mat[i][0],0,n*8);
	}

	m_m = m;
	m_n = n;
	m_snum = 0;
	m_enum = 0;
	m_pnum = 0;

	m_weight = new double[m_m];
	for(i=0;i<m_m;i++)
		m_weight[i] = 1.;
}

matclass::~matclass()
{
	delete_all();
}

matclass::matclass(matclass& mat)
{
	int i;

	mat.getsize(&m_m,&m_n);
	m_mat = new double*[m_m];
	for(i=0;i<m_m;i++)
	{
		m_mat[i] = new double[m_n];
		memcpy(&m_mat[i][0],&mat(i,0),m_n*8);
	}
	m_snum = 0;
	m_enum = 0;
	m_pnum = 0;

	m_weight = new double[m_m];
	for(i=0;i<m_m;i++)
		m_weight[i] = 1.;
}

void matclass::getsize(int *m, int *n)
{
	*m = m_m;
	*n = m_n;
}
```

```
void matclass::delete_all()
{
	int i;

	if(m_m > 0)
	{
		for(i=0;i<m_m;i++)
			delete[] m_mat[i];
		delete[] m_mat;
	}
	delete[] m_weight;
	if(m_enum > 0)
		delete[] m_eigen;
	if(m_snum > 0)
		delete[] m_singular;
	if(m_pnum > 0)
	{
		for(i=0;i<m_m;i++)
			delete[] m_phi[i];
		delete[] m_phi;
		for(i=0;i<m_pnum;i++)
			delete[] m_podc[i];
		delete[] m_podc;
	}
}

void matclass::matrix_sub(matclass a, matclass b)
{
	int i,j,k,m1,m2,n1,n2,k0;

	delete_all();

	a.getsize(&m1,&n1);
	b.getsize(&m2,&n2);

	m_mat = new double*[m1];
	for(i=0;i<m1;i++)
	{
		m_mat[i] = new double[n1];
		memset(&m_mat[i][0],0,n1*8);
	}
	m_m = m1;
	m_n = n1;
	for(i=0;i<m1;i++)
	{
		for(j=0;j<n1;j++)
			m_mat[i][j] = a(i,j) - b(i,j);
	}
```

```
}

void matclass::matrix_add(matclass a, matclass b)
{
	int i,j,k,m1,m2,n1,n2,k0;

	delete_all();

	a.getsize(&m1,&n1);
	b.getsize(&m2,&n2);

	m_mat = new double*[m1];
	for(i=0;i<m1;i++)
	{
		m_mat[i] = new double[n1];
		memset(&m_mat[i][0],0,n1*8);
	}
	m_m = m1;
	m_n = n1;
	for(i=0;i<m1;i++)
	{
		for(j=0;j<n1;j++)
			m_mat[i][j] = a(i,j) + b(i,j);
	}
}

void matclass::matrix_mult_inv(matclass a, matclass b, int ia, int ib)
{
	int i,j,k,m1,m2,n1,n2,k0;
	matclass *tmat;

	a.getsize(&m1,&n1);
	b.getsize(&m2,&n2);

	if(m1 == n1 && m2 == n2 && m1 == n1 && m1 > 0)
	{
		delete_all();

		m_m = m1;
		m_n = n1;
		m_mat = new double*[m_m];
		for(i=0;i<m_m;i++)
		{
			m_mat[i] = new double[m_m];
			memset(&m_mat[i][0],0,m_m*8);
		}
		tmat = new matclass(m_m,m_m);
```

```
if(ia != 0)
{
	a.svd();
	if(ib != 0)
	{
		b.svd();
		for(j=0;j<m_n;j++)
		{
			if(fabsl(a.m_singular[j]) > TOR)
			{
				for(i=0;i<m_m;i++)
					(*tmat)(i,j) = a.m_rmat[j][i] / a.m_singular[j];
			}
		}
		for(i=0;i<m_m;i++)
		{
			for(j=0;j<m_n;j++)
			{
				for(k=0;k<m_n;k++)
					m_mat[i][j] += (*tmat)(i,k) * a.m_lmat[j][k];
			}
		}
		for(i=0;i<m_m;i++)
		{
			memset(&(*tmat)(i,0),0,m_n*8);
			for(j=0;j<m_n;j++)
			{
				for(k=0;k<m_n;k++)
					(*tmat)(i,j) += m_mat[i][k] * b.m_rmat[j][k];
			}
		}
		for(j=0;j<m_n;j++)
		{
			if(fabsl(b.m_singular[j]) > TOR)
			{
				for(i=0;i<m_m;i++)
					(*tmat)(i,j) /= b.m_singular[j];
			}
		}
		for(i=0;i<m_m;i++)
		{
			memset(&m_mat[i][0],0,m_n*8);
			for(j=0;j<m_n;j++)
			{
				for(k=0;k<m_n;k++)
					m_mat[i][j] += (*tmat)(i,k) * b.m_lmat[j][k];
			}
		}
```

```
		}
		else
		{
			for(i=0;i<m_m;i++)
			{
				for(j=0;j<m_n;j++)
				{
					for(k=0;k<m_n;k++)
						(*tmat)(i,j) += a.m_lmat[k][i] * b(k,j);
				}
			}
			for(i=0;i<m_m;i++)
			{
				if(fabsl(a.m_singular[i]) > TOR)
				{
					for(j=0;j<m_n;j++)
						(*tmat)(i,j) /= a.m_singular[i];
				}
			}
			for(i=0;i<m_m;i++)
			{
				for(j=0;j<m_n;j++)
				{
					for(k=0;k<m_n;k++)
						m_mat[i][j] += a.m_rmat[k][i] * (*tmat)(k,j);
				}
			}
		}
	}
	else
	{
		if(ib != 0)
		{
			b.svd();
			for(i=0;i<m_m;i++)
			{
				for(j=0;j<m_n;j++)
				{
					for(k=0;k<m_n;k++)
						(*tmat)(i,j) += a(i,k) * b.m_rmat[j][k];
				}
			}
			for(j=0;j<m_n;j++)
			{
				if(fabsl(b.m_singular[j]) > TOR)
				{
					for(i=0;i<m_m;i++)
						(*tmat)(i,j) /= b.m_singular[j];
```

```
                                        }
                                }
                                for(i=0;i<m_m;i++)
                                {
                                        for(j=0;j<m_n;j++)
                                        {
                                                for(k=0;k<m_n;k++)
                                                        m_mat[i][j] += (*tmat)(i,k) * b.m_lmat[j][k];
                                        }
                                }
                        }
                        else
                        {
                                for(i=0;i<m_m;i++)
                                {
                                        for(j=0;j<m_n;j++)
                                        {
                                                for(k=0;k<m_n;k++)
                                                        m_mat[i][j] += a(i,k) * b(k,j);
                                        }
                                }
                        }
                }

                delete tmat;
        }
}

void matclass::matrix_mult(matclass a, matclass b)
{
        int i,j,k,m1,m2,n1,n2,k0;

        delete_all();

        a.getsize(&m1,&n1);
        b.getsize(&m2,&n2);

        if(n1 > m2)
                k0 = m2;
        else
                k0 = n1;


        m_mat = new double*[m1];
        for(i=0;i<m1;i++)
        {
                m_mat[i] = new double[n2];
                memset(&m_mat[i][0],0,n2*8);
```

```
	}
	m_m = m1;
	m_n = n2;
	for(i=0;i<m1;i++)
	{
		for(j=0;j<n2;j++)
		{
			for(k=0;k<k0;k++)
				m_mat[i][j] += a(i,k) * b(k,j);
		}
	}
}

void matclass::matrix_mult(matclass a, matclass b, int ta, int tb)
{
	int i,j,k,m1,m2,n1,n2,k0;

	delete_all();

	a.getsize(&m1,&n1);
	b.getsize(&m2,&n2);

	if(n1 > m2)
		k0 = m2;
	else
		k0 = n1;

	if(ta == 0)
	{
		if(tb == 0)
		{
			m_mat = new double*[m1];
			for(i=0;i<m1;i++)
			{
				m_mat[i] = new double[n2];
				memset(&m_mat[i][0],0,n2*8);
			}
			m_m = m1;
			m_n = n2;
			for(i=0;i<m1;i++)
			{
				for(j=0;j<n2;j++)
				{
					for(k=0;k<k0;k++)
						m_mat[i][j] += a(i,k) * b(k,j);
				}
			}
		}
```

```
		else
		{
			m_mat = new double*[m1];
			for(i=0;i<m1;i++)
			{
				m_mat[i] = new double[m2];
				memset(&m_mat[i][0],0,m2*8);
			}
			m_m = m1;
			m_n = m2;
			for(i=0;i<m1;i++)
			{
				for(j=0;j<m2;j++)
				{
					for(k=0;k<n1;k++)
						m_mat[i][j] += a(i,k) * b(j,k);
				}
			}
		}
	}
	else
	{
		if(tb == 0)
		{
			m_mat = new double*[n1];
			for(i=0;i<n1;i++)
			{
				m_mat[i] = new double[n2];
				memset(&m_mat[i][0],0,n2*8);
			}
			m_m = n1;
			m_n = n2;
			for(i=0;i<n1;i++)
			{
				for(j=0;j<n2;j++)
				{
					for(k=0;k<m1;k++)
						m_mat[i][j] += a(k,i) * b(k,j);
				}
			}
		}
		else
		{
			m_mat = new double*[n1];
			for(i=0;i<n1;i++)
			{
				m_mat[i] = new double[m2];
				memset(&m_mat[i][0],0,m2*8);
```

```
                        }
                        m_m = n1;
                        m_n = m2;
                        for(i=0;i<n1;i++)
                        {
                                for(j=0;j<m2;j++)
                                {
                                        for(k=0;k<n1;k++)
                                                m_mat[i][j] += a(k,i) * b(k,j);
                                }
                        }
                }
        }
}

double matclass::phi(int m, int n)
{
        if(m_pnum > 0 && n >= 0 && n < m_pnum)
                return m_phi[m][n];
        else
                return 0.;
}

double matclass::podc(int m, int n)
{
        if(m_pnum > 0 && m >= 0 && m < m_pnum)
                return m_podc[m][n];
        else
                return 0.;
}

double matclass::lsq(matclass *b, matclass *c)
{
        int i,j,k,mb,nb;
        double *vec,tempd1,tempd2,ave,std;

        svd();
        b->getsize(&mb,&nb);

        for(i=0;i<m_n;i++)
        {
                for(j=0;j<nb;j++)
                {
                        (*c)(i,j) = 0.;
                        for(k=0;k<m_m;k++)
                                (*c)(i,j) += m_lmat[k][i] * (*b)(k,j);
                }
        }
```

```
for(i=0;i<m_n;i++)
{
	if(fabsl(m_singular[i]) > TOR)
	{
		for(j=0;j<nb;j++)
			(*c)(i,j) /= m_singular[i];
	}
}
vec = new double[m_n];
for(j=0;j<nb;j++)
{
	for(i=0;i<m_n;i++)
	{
		vec[i] = 0.;
		for(k=0;k<m_n;k++)
			vec[i] += m_rmat[k][i] * (*c)(k,j);
	}
	for(i=0;i<m_n;i++)
		(*c)(i,j) = vec[i];
}
delete[] vec;

vec = new double[nb];
ave = 0.;
std = 0.;
for(i=0;i<m_m;i++)
{
	for(j=0;j<nb;j++)
	{
		vec[j] = (*b)(i,j);
		for(k=0;k<m_n;k++)
			vec[j] -= m_mat[i][k] * (*c)(k,j);
	}
	tempd1 = 0.;
	tempd2 = 0.;
	for(j=0;j<nb;j++)
	{
		tempd1 += (*b)(i,j) * (*b)(i,j);
		tempd2 += vec[j] * vec[j];
	}
	ave += tempd2 / tempd1;
	std += tempd2 * tempd2 / tempd1 / tempd1;
}
std = std/(m_m - 1.) - ave*ave/(m_m*(m_m - 1.));
std = sqrt(std);
ave /= m_m;

delete[] vec;
```

```
    return ave;
}
```