基于分层技术的快速最大方差展开算法及其在过程监测中的应用

魏驰航[1]，宋执环[1]，陈荣辉[2]

（1. 浙江大学控制科学与工程学院，杭州 310027;
2. 中原大学工学院，台湾桃园 32023）

摘要：提出一种针对大规模数据的基于分层技术的快速最大方差展开（fast maximum variance unfolding，FMVU）算法。该算法通过分层技术求取近似核来显著减小计算复杂度和空间要求，同时最小化所牺牲的准确性。数据的局部特征和稀疏性保证了 FMVU 分层技术的有效实施。本文给出了 FMVU 算法的数学结构，然后定量地推导了精确性、计算复杂度和空间要求。通过一个数学例子和连续搅拌釜式加热器（continuous stirred tank heater，CSTH）过程验证了所提算法的可行性与有效性。

关键词：自动控制技术；过程监测；快速最大方差展开；分层技术；大规模数据

中图分类号：TP277　　**文献标识码：**A　　**文章编号：**1674-2850(2018)16-1651-08

Fast maximum variance unfolding algorithm and its application for process monitoring based on hierarchical technique

WEI Chihang[1], SONG Zhihuan[1], CHEN Junghui[2]

(1. *College of Control Science and Engineering*, *Zhejiang University*, *Hangzhou* 310027, *China*;
2. *College of Engineering*, *Chung Yuan Christian University*, *Taoyuan*, *Taiwan* 32023, *China*)

Abstract: In this paper, a learning framework for fast maximum variance unfolding (FMVU) algorithm based on hierarchical technique is proposed to learn a scalable approximate kernel that helps to reduce computational complexity and storage requirements for large-scale datasets, as well as minimize the sacrificed accuracy. The hierarchical construction of FMVU from the collected data is achieved by the good localization characteristics and sparsity of data. The mathematical framework for the development of FMVU algorithm and quantitative derivation on the accuracy, computational complexity as well as storage requirements are presented in this paper. The feasibility and efficiency of the proposed method is illustrated through a numerical case and the continuous stirred tank heater (CSTH) process.

Key words: autocontrol technology; process monitoring; fast maximum variance unfolding; hierarchical technique; large-scale dataset

0 引言

近些年，厂级的过程监测吸引了越来越多研究者的关注[1]。分布式控制器的广泛应用使采集大规模数据成为可能，也推动了基于数据的多变量统计过程监测的发展[2]。由于厂级过程广泛存在内在的质量、能量等平衡的约束，其数据往往存在非常严重的变量冗余现象。因此，基于数据的多变量统计过程监测需要首先将数据降维到低维空间，然后在低维空间中进行过程监测。

基金项目：高等学校博士学科点专项科研基金（20130101110138）

作者简介：魏驰航（1990—），男，博士研究生，主要研究方向：过程监测、流行学习

通信联系人：宋执环，教授，主要研究方向：复杂工程系统安全监测与故障诊断、工业大数据分析与建模. E-mail: songzhihuan@zju.edu.cn

在多种多样的基于数据的多变量统计过程监测算法中，主成分分析可能是最为人熟知并得到广泛应用的算法[3]。该算法最明显的缺点是不能处理非线性数据。因此，有学者提出了核主成分分析[3]。但是由于没有有效且统一的选取核函数的方法，核主成分分析算法的效果得不到保证[4]。在此背景下，有学者提出了最大方差展开（maximum variance unfolding，MVU）算法[4]，并且该算法已被证明是一种在过程监测领域中非常有效的非线性数据降维方法。其最大特点是可以自主从建模数据中学习出来一个核，而不是像核主成分分析那样由使用者人为地指定一个核，以此来更好地适应特定数据。然而在实际使用中，由于计算复杂度和空间要求的限制，MVU算法只适用于小规模数据[5]。

为解决这个问题，本文提出基于分层技术的 FMVU 算法。该算法根据数据分布和稀疏性对不同的数据点采取不同的处理方式，而不是像传统方法平等地处理所有数据点。具体来说，对于传统的 MVU 算法，在离线建模阶段，只有一个 $N\times N$ 的核矩阵被学习出来（N 为建模点的数量），然后在此基础上进行数据降维；在在线监测阶段，新数据点的降维结果要经过与所有 N 个建模数据点间的两两计算得出。而对于本文所提出的算法，首先根据数据点间的欧式距离将建模数据分为 P 类，然后选择 P 个代表点；将传统的 MVU 算法分别作用于这 P 个代表点与 P 类上，得到 P+1 个核；最终通过所提的技术将这些核全部融合在一起得到一个近似核。这将显著降低计算复杂度与空间要求。

1 MVU 算法回顾

MVU算法可以自主学习到一个可以将数据在低维度空间展开的核，更准确地说是优化出一个核[4]。假设 $X=\{\boldsymbol{x}_n\}_{n=1}^{N}$（$\boldsymbol{x}_n\in\mathbb{R}^D$）为建模数据，$Y=\{\boldsymbol{y}_n=\boldsymbol{\Phi}(\boldsymbol{x}_n)\}_{n=1}^{N}$（$\boldsymbol{y}_n\in\mathbb{R}^d$）为降维后的空间，其中$N$为建模点的数量，$D$为原空间维度，$d$为降维空间的维度（$d<D$）。目标函数的目的是在低维空间展开数据，因此将目标函数设置为最大化低维空间两两数据点间距离的平方和，即

$$\max \Gamma=\max\frac{1}{2N}\sum_{i,j}\|\boldsymbol{\Phi}(\boldsymbol{x}_i)-\boldsymbol{\Phi}(\boldsymbol{x}_j)\|^2. \tag{1}$$

与此同时，此优化还应该加上两个约束：局部等距约束和中心化约束。

局部等距约束的目的是在核空间保持数据流行的局部结构。假设 $\boldsymbol{S}\in\mathbb{R}^{N\times N}$ 为一个二元邻接矩阵，其数据代表了 $\boldsymbol{x}_i$ 与 $\boldsymbol{x}_j$ 是否为临近点。此约束具体定义如下：

$$\left\|\boldsymbol{\Phi}(\boldsymbol{x}_i)-\boldsymbol{\Phi}(\boldsymbol{x}_j)\right\|^2=\left\|\boldsymbol{x}_i-\boldsymbol{x}_j\right\|^2=D_{i,j},\quad \text{对所有 } S_{i,j}=1 \text{或} \left[\boldsymbol{S}^{\mathrm{T}}\boldsymbol{S}\right]_{i,j}=1. \tag{2}$$

中心化约束的目的是去除最终优化结果的一个自由度：

$$\sum_i\boldsymbol{\Phi}(\boldsymbol{x}_i)=0. \tag{3}$$

由于上述优化包含二次项，因此它不是凸优化。这里为简便计算，引入 $K_{i,j}=\boldsymbol{\Phi}^{\mathrm{T}}(\boldsymbol{x}_i)\boldsymbol{\Phi}(\boldsymbol{x}_j)$. 这样目标函数（1）和约束（2）、（3）均可以表达为 $K_{i,j}$ 的形式，也就得到了最终的优化形式：

$$\begin{aligned}&\max_{\boldsymbol{K}}\ \mathrm{tr}(\boldsymbol{K}),\\ \text{s.t.}\quad &\boldsymbol{K}\succcurlyeq 0,\\ &\sum_{i,j}K_{i,j}=0,\\ &K_{i,i}+K_{j,j}-2K_{i,j}=D_{i,j},\ \text{对所有}S_{i,j}=1\text{或}\left[\boldsymbol{S}^{\mathrm{T}}\boldsymbol{S}\right]_{i,j}=1,\end{aligned} \tag{4}$$

其中，半正定矩阵 $\boldsymbol{K}$ 为学习出来的核。目前有很多算法工具来求解这个问题，如基于C语言库的半正定规划求解方法（C library for semi-definite programming，CSDP）[5]。 需要特别指出的是，这个优化的解是全局最优的。

2 FMVU 算法

随着近几十年的“数据爆炸”，更多的操作数据需要被分析。然而在这些数据中发掘真正有用的信息是很有挑战性的。MVU算法就是这样一种可以有效提取信息的降维工具。假设N为建模点的数量，C为约束的数量，MVU算法所需的计算复杂度为$O^{\mathrm{C}}=O(N^3+C^3)$[5]，而空间要求为$O^{\mathrm{S}}=O(N^2+C^2)$[5]. 可以明显看出传统的MVU算法只能用来对小规模数据降维，这大大限制了MVU算法的使用范围。直觉上来看，直接减小N是解除这个“诅咒”的有效办法。更具体地来说，算法的目标是使用更少的点去学习一个近似的核，同时不能损失太多的精确度。精确度可以如下定量度量：

$$\Theta=\sum_{i=1}^{N}\sum_{j=1}^{N}(K_{i,j}-\hat{K}_{i,j})^2=\|\boldsymbol{K}-\hat{\boldsymbol{K}}\|_2^2. \tag{5}$$

2.1 FMVU 算法理论推导

FMVU 算法建立在如下假设上：所得到的核是平滑的，在建模数据中距离相近的点所对应的核数据是相似的。这个假设对于真实世界来说是普适的，并且意味着在核学习时可以使用相近点的均值点来代替这些点。因此，可以将建模数据分成若干子类，每个子类内数据点间的相似性远远大于不同子类间的相似性。FMVU 算法是一个两层的“树”状结构，包括一个“全局层”和一个“局部层”。在全局层中，首先将原始的建模数据 $X=\{\boldsymbol{x}_n\}_{n=1}^{N}$ 均等分为 P 类，每一类包含 $N_P=N/P$ 个数据点，进而计算出每一类的代表性点 $\boldsymbol{x}_p^{(1)}$：

$$\boldsymbol{x}_p^{(1)}=\frac{1}{N_P}\sum_{j\in\Upsilon(p)}\boldsymbol{x}_j\,, \tag{6}$$

其中，$\Upsilon(p)=\{H^p+1,\cdots,H^p+N_P\}$（$H^p=(p-1)N_P$，$p=1,2,\cdots,P$）代表了第 p 类所含建模点的序号。将所有的代表性点记为 $X^{(1)}=\{\boldsymbol{x}_i^{(1)}\}_{i=1}^{P}$，并且使用式（4）中的优化学习到核矩阵 $\boldsymbol{K}^{(1)}\in\mathbb{R}^{P\times P}$. $\boldsymbol{K}_{i,j}^{(1)}=\kappa(\boldsymbol{x}_i^{(1)},\boldsymbol{x}_j^{(1)})$ 代表类 i 与类 j 间的核值。

由于 $\boldsymbol{x}_i^{(1)}$ 和 $\boldsymbol{x}_j^{(1)}$ 是类 i 与类 j 的代表性点，只需要简单重复 $\kappa(\boldsymbol{x}_i^{(1)},\boldsymbol{x}_j^{(1)})$ 来填满子矩阵 $\hat{\boldsymbol{K}}_{i,j}^{(1)}\in\mathbb{R}^{N_P\times N_P}$，

$$\hat{\boldsymbol{K}}_{i,j}^{(1)}=\begin{bmatrix}\kappa(\boldsymbol{x}_i^{(1)},\boldsymbol{x}_j^{(1)}) & \cdots & \kappa(\boldsymbol{x}_i^{(1)},\boldsymbol{x}_j^{(1)})\\ \vdots & & \vdots\\ \kappa(\boldsymbol{x}_i^{(1)},\boldsymbol{x}_j^{(1)}) & \cdots & \kappa(\boldsymbol{x}_i^{(1)},\boldsymbol{x}_j^{(1)})\end{bmatrix}, \tag{7}$$

其中，$i,j=1,2,\cdots,P$；$\kappa(\cdot\,,\cdot)$ 代表相应的核方程。在每一块中，所有的元素均相同。图 1 中的近似核矩阵（$\hat{\boldsymbol{K}}$，$\hat{\boldsymbol{K}}^{(1)}\in\mathbb{R}^{N\times N}$）定义为

$$\hat{\boldsymbol{K}}=\hat{\boldsymbol{K}}^{(1)}=\begin{bmatrix}\hat{\boldsymbol{K}}_{1,1}^{(1)} & \cdots & \hat{\boldsymbol{K}}_{1,j}^{(1)} & \cdots & \hat{\boldsymbol{K}}_{1,P}^{(1)}\\ \vdots & & \vdots & & \vdots\\ \hat{\boldsymbol{K}}_{i,1}^{(1)} & \cdots & \hat{\boldsymbol{K}}_{i,j}^{(1)} & \cdots & \hat{\boldsymbol{K}}_{i,P}^{(1)}\\ \vdots & & \vdots & & \vdots\\ \hat{\boldsymbol{K}}_{P,1}^{(1)} & \cdots & \hat{\boldsymbol{K}}_{P,j}^{(1)} & \cdots & \hat{\boldsymbol{K}}_{P,P}^{(1)}\end{bmatrix}$$

$$=\underbrace{\begin{bmatrix}\hat{\boldsymbol{K}}_{1,1}^{(1)} & \cdots & \boldsymbol{0} & \cdots & \boldsymbol{0}\\ \vdots & & \vdots & & \vdots\\ \boldsymbol{0} & \cdots & \hat{\boldsymbol{K}}_{i,j}^{(1)} & \cdots & \boldsymbol{0}\\ \vdots & & \vdots & & \vdots\\ \boldsymbol{0} & \cdots & \boldsymbol{0} & \cdots & \hat{\boldsymbol{K}}_{P,P}^{(1)}\end{bmatrix}}_{\hat{\boldsymbol{K}}_D^{(1)}}+\underbrace{\begin{bmatrix}\boldsymbol{0} & \cdots & \hat{\boldsymbol{K}}_{1,j}^{(1)} & \cdots & \hat{\boldsymbol{K}}_{1,P}^{(1)}\\ \vdots & & \vdots & & \vdots\\ \hat{\boldsymbol{K}}_{i,1}^{(1)} & \cdots & \boldsymbol{0} & \cdots & \hat{\boldsymbol{K}}_{i,P}^{(1)}\\ \vdots & & \vdots & & \vdots\\ \hat{\boldsymbol{K}}_{P,1}^{(1)} & \cdots & \hat{\boldsymbol{K}}_{P,j}^{(1)} & \cdots & \boldsymbol{0}\end{bmatrix}}_{\hat{\boldsymbol{K}}_O^{(1)}}. \tag{8}$$

全局层的近似核的精确度 $\Theta^{(1)}$ 可以通过如下计算得到：

$$\Theta^{(1)}=\sum_{i=1}^{N}\sum_{j=1}^{N}(K_{i,j}-\hat{K}_{i,j})^2=\|\boldsymbol{K}-\hat{\boldsymbol{K}}^{(1)}\|_2^2$$

$$=\sum_{i=1}^{P}\sum_{j=1}^{P}\sum_{i_1=H^i+1}^{H^i+N_P}\sum_{j_1=H^j+1}^{H^j+N_P}\boldsymbol{O}_{i,j}^{U}\left[\kappa'(\boldsymbol{x}_{i_1},\boldsymbol{x}_{j_1})(\boldsymbol{e}_{i_1-H^i}^{i}+\boldsymbol{e}_{j_1-H^j}^{j})\right]^2, \tag{9}$$

其中，$H^i=(i-1)N_P$；$\kappa'(\boldsymbol{x}_{i_1},\boldsymbol{x}_{j_1})$ 为 $\kappa(\boldsymbol{x}_{i_1},\boldsymbol{x}_{j_1})$ 的偏微分；$\boldsymbol{e}_{i_1-H^i}^{i}$ 和 $\boldsymbol{e}_{j_1-H^j}^{j}$ 通过泰勒级数展开得到。

在上述推导中，矩阵 $\boldsymbol{O}^U=\left[\boldsymbol{O}_{i,j}^{U}\right]$ 为一个全 1 矩阵，

$$\boldsymbol{O}^U=\begin{bmatrix}\boldsymbol{1} & \cdots & \boldsymbol{1}\\ \vdots & & \vdots\\ \boldsymbol{1} & \cdots & \boldsymbol{1}\end{bmatrix}, \tag{10}$$

$\boldsymbol{e}_{i_1}^{i}=\boldsymbol{x}_{i_1-H^i}-\boldsymbol{x}_i^{(1)}$ 和 $\boldsymbol{e}_{j_1}^{i}=\boldsymbol{x}_{j_1-H^j}-\boldsymbol{x}_j^{(1)}$ 代表真实数据点（$\boldsymbol{x}_{i_1-H^i}$ 和 $\boldsymbol{x}_{j_1-H^j}$）与所对应类别代表性点（$\boldsymbol{x}_i^{(1)}$ 和 $\boldsymbol{x}_j^{(1)}$）之间的差值，其中，$i_1=H^i+1,\cdots,H^i+N_P$，$j_1=H^j+1,\cdots,H^j+N_P$，$i,j=1,2,\cdots,P$. 随着 P 的变大，N_P 会变小，进而 $\boldsymbol{e}_{i_1}^{i}$ 和 $\boldsymbol{e}_{j_1}^{i}$ 会变小，精确度上升。然而计算复杂度 O^{C} 和空间要求 O^{S} 限制了 P 的最大值。为在有限的计算资源的前提下进一步提高准确性，每一类内的点也被引入到算法中。如图 1 所示，在类 i 内使用 $X^{(2),i}=\left\{\boldsymbol{x}_j\right\}_{j\in\Upsilon(i)}$ 来学习到核矩阵 $\boldsymbol{K}^{(2),i}\in\mathbb{R}^{N_P\times N_P}$. 即

$$\boldsymbol{K}^{(2),i}=\begin{bmatrix}\kappa(\boldsymbol{x}_{i_1},\boldsymbol{x}_{i_1}) & \cdots & \kappa(\boldsymbol{x}_{i_1},\boldsymbol{x}_{i_1+N_P})\\ \vdots & & \vdots\\ \kappa(\boldsymbol{x}_{i_1+N_P},\boldsymbol{x}_{i_1}) & \cdots & \kappa(\boldsymbol{x}_{i_1+N_P},\boldsymbol{x}_{i_1+N_P})\end{bmatrix}, \tag{11}$$

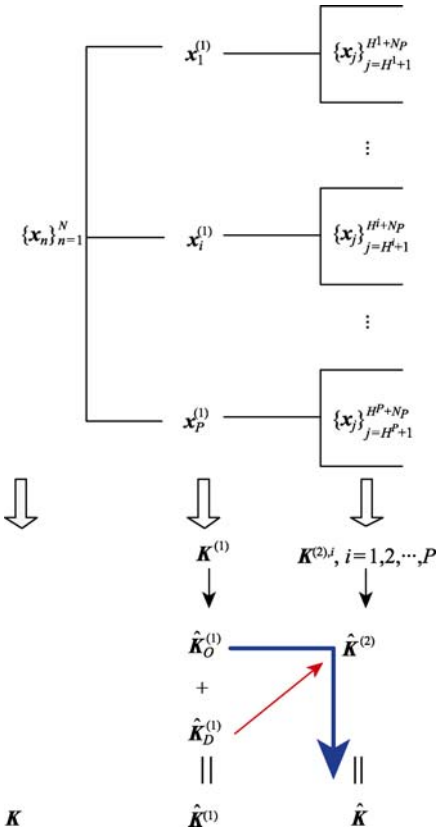

图 1 FMVU 算法的分解与组合

Fig. 1 Decomposition and combination of FMVU algorithm

其中，$i_1 = H^i + 1$（$i = 1,2,\cdots,P$）. 由于 $\boldsymbol{K}^{(2),i} \in \mathbb{R}^{N_P \times N_P}$ 代表类 i 的核矩阵，只需将这个矩阵填充到子矩阵 $\boldsymbol{K}^{(2)}$ 的主对角块处。因此，局部层的近似核矩阵（$\hat{\boldsymbol{K}}^{(2)} \in \mathbb{R}^{N \times N}$）定义为

$$\hat{\boldsymbol{K}}^{(2)} = \begin{bmatrix} \hat{\boldsymbol{K}}_{1,1}^{(2)} & \cdots & \mathbf{0} & \cdots & \mathbf{0} \\ \vdots & & \vdots & & \vdots \\ \mathbf{0} & \cdots & \hat{\boldsymbol{K}}_{i,i}^{(2)} & \cdots & \mathbf{0} \\ \vdots & & \vdots & & \vdots \\ \mathbf{0} & \cdots & \mathbf{0} & \cdots & \hat{\boldsymbol{K}}_{P,P}^{(2)} \end{bmatrix}. \tag{12}$$

接着可以得到 $\hat{\boldsymbol{K}}$：

$$\hat{\boldsymbol{K}} = \hat{\boldsymbol{K}}_O^{(1)} + \hat{\boldsymbol{K}}^{(2)}. \tag{13}$$

全局层的近似核的精确度 $\Theta^{(2)}$ 可以通过如下计算得到：

$$\begin{aligned} \Theta^{(2)} &= \sum_{i=1}^{N}\sum_{j=1}^{N}(K_{i,j} - \hat{K}_{i,j})^2 = \| \boldsymbol{K} - \hat{\boldsymbol{K}}_O^{(1)} - \hat{\boldsymbol{K}}^{(2)} \|_2^2 \\ &= \sum_{i=1}^{P}\sum_{j=1}^{P}\sum_{i_1=H^i+1}^{H^i+N_P}\sum_{j_1=H^j+1}^{H^j+N_P} \boldsymbol{O}_{i,j}^{L}\left[\kappa(\boldsymbol{x}_{i_1}, \boldsymbol{x}_{j_1})(\boldsymbol{e}^i_{i_1-H^i} + \boldsymbol{e}^j_{j_1-H^j})\right]^2, \end{aligned} \tag{14}$$

其中，

$$\boldsymbol{O}^L = \begin{bmatrix} \mathbf{0} & \mathbf{1} & \mathbf{1} & \cdots & \mathbf{1} \\ \mathbf{1} & \mathbf{0} & \mathbf{1} & \cdots & \mathbf{1} \\ \vdots & \vdots & \vdots & & \vdots \\ \mathbf{1} & \mathbf{1} & \mathbf{1} & \cdots & \mathbf{0} \end{bmatrix}, \tag{15}$$

0 和 **1** 代表全 0 与全 1 矩阵，**0** 和 **1** 大小均为 $N_P \times N_P$. 由于 $\boldsymbol{O}^L$ 的对角块矩阵元素均为 0，很明显可以得到 $\Theta^{(2)} \leqslant \Theta^{(1)}$.

2.2 计算复杂度与空间约束

上文已经提到，假设 N 为建模点的数量，C 为约束的数量，MVU 算法所需的计算复杂度为 $O^{\mathrm{C}}=O(N^3+C^3)$，而空间要求为 $O^{\mathrm{S}}=O(N^2+C^2)$. 由于 C 可以认为约等于 kN，其中 k 为临近点的数量，传统 MVU 算法的计算复杂度为

$$O_{\mathrm{MVU}}^{\mathrm{C}} = O(N^3 + C^3) = O((1+k^3)N^3). \tag{16}$$

FMVU 算法的计算复杂度为

$$O_{\mathrm{FMVU}}^{\mathrm{C}} = O(P^3 + C_1^3) + P \times O(N_P^3 + C_2^3) = O((1+k_1^3)P^3) + O(\frac{(1+k_2^3)N^3}{P^2}), \tag{17}$$

其中，C_1 和 C_2 分别为全局层与局部层的约束数量；k_1 和 k_2 分别为全局层与局部层的临近点数量。由于 $P \ll N$，很容易得到 $O_{\mathrm{FMVU}}^{\mathrm{C}} \ll O_{\mathrm{MVU}}^{\mathrm{C}}$. 同样地，FMVU 算法的空间要求 $O_{\mathrm{FMVU}}^{\mathrm{S}}$ 远比传统 MVU 算法的空间要求 $O_{\mathrm{MVU}}^{\mathrm{S}}$ 小，即 $O_{\mathrm{FMVU}}^{\mathrm{S}} \ll O_{\mathrm{MVU}}^{\mathrm{S}}$.

2.3 基于 FMVU 算法的过程监测

将 $\hat{\boldsymbol{K}}$ 进行特征值分解，

$$\lambda_l \boldsymbol{\alpha}_l = \hat{\boldsymbol{K}}\boldsymbol{\alpha}_l = \left(\hat{\boldsymbol{K}}_O^{(1)} + \hat{\boldsymbol{K}}^{(2)}\right)\boldsymbol{\alpha}_l \,, \tag{18}$$

结果说明得分 $\hat{\boldsymbol{y}}_n$ 是两层中核矩阵得分的和。因此得分 $\hat{\boldsymbol{y}}_n$ 可由下式近似计算得到：

$$\hat{\boldsymbol{y}}_n = \hat{\boldsymbol{y}}_n^{(1)} + \hat{\boldsymbol{y}}_n^{(2)} = \boldsymbol{y}_p^{(1)} + \boldsymbol{y}_{n-H}^{(2),p} . \tag{19}$$

类似于基于 MVU 算法的过程监测，本文使用经典的 T^2 与 Q 统计量来监测系统运行状态是否异常。

3 仿真

3.1 基于 FMVU 算法的过程监测

此数值仿真系统来源于一个有缺口的圆环，

$$\begin{aligned} x_1 &= \sin(t) + e_1, \\ x_2 &= \cos(t) + e_2, \end{aligned} \tag{20}$$

其中，e_1和e_2为独立的随机噪声，服从高斯分布 $N(0,0.01)$；t 为输入参数，服从均匀分布 $[0.1\pi,1.9\pi]$. e_1、e_2 与 t 均不可测，而只有 x_1 和 x_2 可测得。

设置该数值仿真的初衷在于评价 FMVU 算法所学习出的近似核的精确度。为得到传统 MVU 算法所学习出的精确的核，建模点的数量被限制在很低的水平。如图 2 所示，320 个采样点被选作建模数据。传统 MVU 算法中通过直接使用全部的建模数据来学习出核 $\boldsymbol{K} \in \mathbb{R}^{320\times320}$. 而对 FMVU 算法，在全局层中，数据被分为 P=16 类，每类含 N_P=20 个数据点，只有一个核 $\boldsymbol{K}^{(1)} \in \mathbb{R}^{16\times16}$ 被通过图 3a 的点学习出来；在局部层中，分别使用图 3b 中的点学习出核 $\boldsymbol{K}^{(2),i} \in \mathbb{R}^{20\times20}$，$i=[1,\cdots,16]$. 所有算法中临近点的设置均遵从文献[4]。表 1 给出了上述算法的所有参数。如表 1 所示，FMVU 算法的速度远远比传统 MVU 算法快。表 1 中同时给出了 FMVU 算法每层的精确度，并且确认了精确度在局部层中有所提升（$\Theta^{(1)} > \Theta^{(2)}$）。仿真结果证实了 2 节中的理论分析。

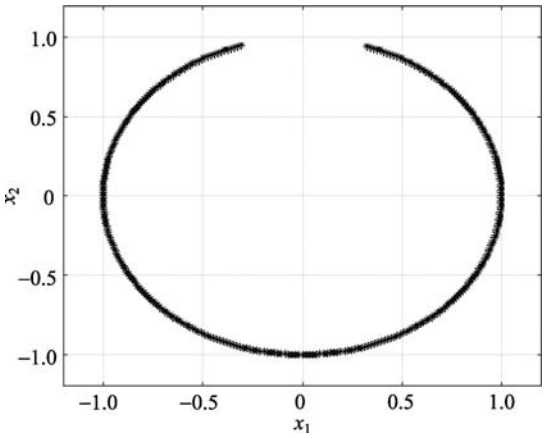

图 2 数值仿真中的建模样本

Fig. 2 Training samples in the numerical simulation

3.2 CSTH 过程

这个例子主要考察传统 MVU 算法与 FMVU 算法在 CSTH 过程上的过程监测效果。CSTH 过程最初由文献[6]提出，使用 PI 控制器。一共选择了 6 个变量进行过程监测：蒸汽阀门、冷水阀门、控制器输出信号、液位、冷水流量和温度。在正常工况下收集的 600 个数据点被选为建模数据。另外产生了 4 个测试数据集，依次为正常工况、液位随机改变故障工况、冷水阀门随机改变故障工况和冷水流量阶跃改变故障工况，分别命名为 IDV0、IDV1、IDV2 和 IDV3，每个测试数据集都含有 600 个数据点。所有故障均在第 201 个点被引入。表 2 给出了所有的故障检测率。为更直观地呈现结果，图 4 分别给出了传统 MVU 算法和 FMVU 算法在 IDV1 数据集上的具体检测结果。由检测结果可以很容易地看出，FMVU 算法与传统 MVU 算法在过程监测领域的表现基本相同，但是 FMVU 算法所用时间明显减少。

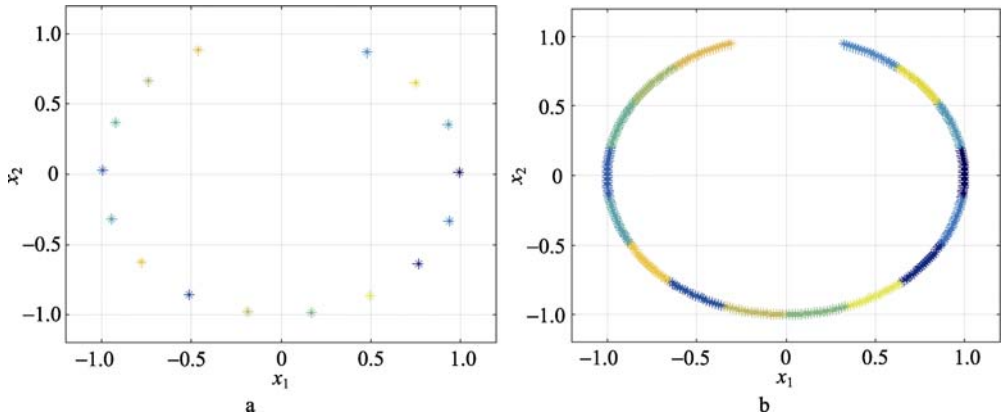

图 3 FMVU 算法全局层（a）和局部层（b）的建模样本

Fig. 3 Training samples of FMVU algorithm in global level (a) and local level (b)

表 1 数值仿真的参数、训练时间与精确度

Tab. 1 Parameters, training time and accuracy of numerical simulation

算法	层	学习核所用的点的数量	临近点数	训练时间/s	Θ 的均值
MVU	—	360	2	5.692 3	0
FMVU	全局层	16	2	0.204 7	$\Theta^{(1)}/320^2 = 0.512\ 0$
	局部层	20	2	1.125 3	$\Theta^{(2)}/320^2 = 0.466\ 7$

表 2 传统 MVU 算法和 FMVU 算法的检测率

Tab. 2 Detection rate of traditional MVU algorithm and FMVU algorithm

故障号	MVU		FMVU	
	T^2	Q	T^2	Q
IDV0	0.00	0.01	0.01	0.02
IDV1	0.75	0.81	0.73	0.83
IDV2	0.45	0.68	0.45	0.67
IDV3	1.00	1.00	1.00	1.00

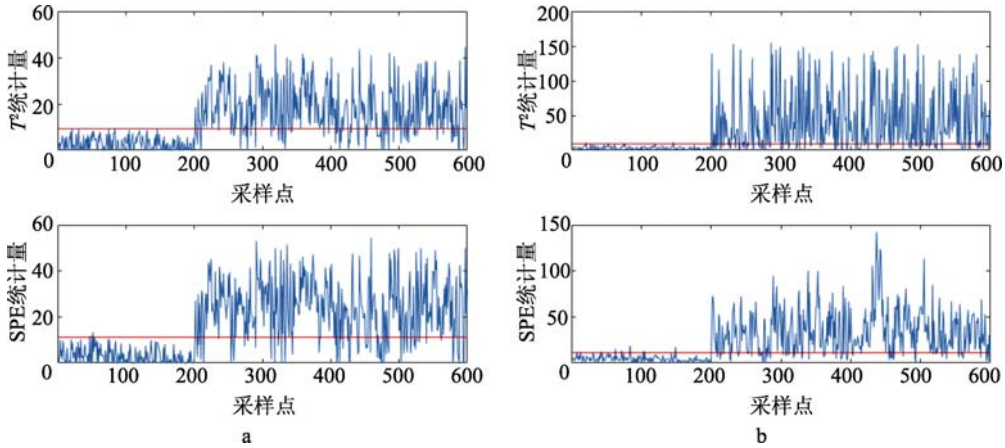

图 4 利用传统 MVU 算法（a）和 FMVU 算法（b）检测 IDV1 数据集的结果

Fig. 4 Detailed detection results of IDV1 dataset using traditional MVU algorithm (a) and FMVU algorithm (b)

4 结论

本文提出了一种基于分层技术的FMVU算法，该算法可以在保证精确度、保证过程监测效果的同时显著降低计算复杂度和空间要求，以此来适应大规模数据的应用。本文通过一个数学例子和CSTH过程验证了所提算法的有效性。FMVU算法将建模数据切分为多类并将这些类分配到局部层，同时在每一类内选取一个代表性点来组成全局层。这样使得传统MVU算法仅作用于较小的数据集上，以此来显著降低计算复杂度和空间要求。如果每一类仍然包含有太多的数据点以至于计算机无法处理，可以在每一类内进行切分，这样就在一个类内产生了一个新的两层树。这个切分过程可以一直进行下去直到每一类所含的点数都可以被计算机直接处理。

[参考文献] (References)

[1] VENKATASUBRAMANIAN V, RENGASWAMY R, YIN K, et al. A review of process fault detection and diagnosis. Part I: quantitative model-based methods[J]. Computers & Chemical Engineering, 2003, 27: 293-311.

[2] QIN S J. Statistical process monitoring: basics and beyond[J]. Journal of Chemometrics, 2003, 17(8-9): 480-502.

[3] LEE J M, YOO C K, CHOI S W, et al. Nonlinear process monitoring using kernel principal component analysis[J]. Chemical Engineering Science, 2004, 59(1): 223-234.

[4] WEINBERGER K Q, SAUL L K. Unsupervised learning of image manifolds by semidefinite programming[J]. International Journal of Computer Vision, 2006, 70(1): 77-90.

[5] BORCHERS B. CSDP, a C library for semidefinite programming[J]. Optimization Methods and Software, 1999, 11(1-4): 613-623.

[6] THORNHILL N F, PATWARDHAN S C, SHAH S L. A continuous stirred tank heater simulation model with applications[J]. Journal of Process Control, 2008, 18(3-4): 347-360.