解大型结构特征值问题的同时迭代的一个新算法

飞机结构强度研究所　刘国光　李军杰

摘　要

本文提出了同时迭代法的一种变型算法，用来求解大型结构动力分析中的广义特征值问题。使用了E_k子空间和特征方向的概念，证明了该算法的收敛率，并讨论了如何在计算机上实现的某些细节问题。通过统计每步迭代所需运算量和大量的实际计算，表明该算法与当前几种常用的同时迭代法或子空间迭代法相比，收敛率相同，但减少了每步迭代所需的计算量，提高了计算效率。

一、前　言

结构动力分析中的特征值问题，经过奇异性处理（譬如原点移位）后，其形式为

$$Kx=\lambda Mx \tag{1}$$

其中K和M为n阶实对称正定、稀疏矩阵。现设系统（1）的特征值为λ_1，λ_2，……，λ_n，并且有

$$0<\lambda_1\leqslant\lambda_2\leqslant\cdots\cdots\leqslant\lambda_n \tag{2}$$

相应的特征向量为x_1，x_2，……，x_n。

对于大型结构，系统（1）的阶数可高达几千阶。在这种情况下，应用K和M的稀疏性将是十分必要的。因此，同时迭代算法已成为解系统（1）的部分低阶特征对（特征值和相应的特征向量）的强有力的方法之一。当前由Rutishauer，Reinsch，Mc Cormick，Bathe和Nicolai等人提出的同时迭代的变型算法较为流行。其中前两种算法是对M进行Cholesky分解，然后再把系统（1）转换成标准特征值问题$Ay=\lambda y$求解。即使不考虑由M分解而产生的误差影响，用这样的算法求解系统（1）部分特征值问题也是不利的(见表4)。这不仅由于M分解本身将花费一定的机时，而且由于分解“填充”将会产生新的非零元素，于是使得以后各迭代步增加一定的计算量，从而影响了计算效率。

另外，Reinsch和Nicolai的算法中使用了Gram-Schmidt直交化过程，因而可以把已收敛的试向量固定住，不再参加以后的迭代。这种处理技巧[2]通常称为“降维”处理。当待求特征对的个数比较多时，采用降维处理将会大大提高计算效率（见表6）。

1980年10月收到。

本文给出一种新的变型算法，与上述算法相比，除了不分解 M 和可采用降维处理外，还具有进一步减少了每步迭代所花费的计算量这一重要的特点。

下一节我们将简单地介绍系统（1）的 E_k 子空间概念，为证明本文算法的收敛率提供理论基础。第三节介绍算法并叙述在计算机上实现方面的某些细节问题。关于方法比较与数值检验结果将分别在第四、五两节中给出。

用大写字母表示矩阵。特别地，用 R 表示上三角矩阵，用 r_{ij} 表示它的（i，j）元素。用 D 表示对角矩阵，用 d_{ij} 表示它的第 j 个对角线元素。用 X，Y，Z 和 S 表示 $n\times p$ 向量矩阵，并分别用 x_j，y_j，z_j，s_j 表示它们的第 j 列向量。用大写字母 O 表示同阶小量。

二、系统(1)的 E_k 子空间

设 $x_1^{(0)}$，$x_2^{(0)}$，……，$x_p^{(0)}$ 为 n 维空间中任一与线性无关的向量组。如果令 E_0 为它们所张成的子空间，那么系统（1）的 E_k 子空间则可定义如下：

$$E_k=\{x\mid Kx=My,\quad y\in E_{k-1}\} \tag{3}$$

由 K 和 M 的对称正定性可知，对任意的 k，E_k 子空间的维数都为 p，而且 $\lim\limits_{k\to\infty}E_k$ 存在，并为系统（1）的一个不变子空间。如果它是由系统（1）前 p 个低阶特征向量 x_1，x_2，……，x_p 所张成的，那么就称 E_k 为稳定收敛。

理论上，在稳定收敛的情况下，在 E_k 上总可以找到称之为最佳方向的 p 个向量 u_1，……，u_p[1]，并有

$$\|u_j-x_j\|=O(q_j^k) \tag{4}$$

其中

$$q_j=\lambda_j/\lambda_{p+1} \tag{5}$$

但由于最佳方向无法计算，所以同时迭代的各种算法都是在 E_k 上通过一组已知的 p 个向量来计算一组新的向量 $x_1^{(k)}$，……，$x_p^{(k)}$，使得当 k 充分大时，它们渐近地趋向于最佳方向。于是有

$$\|x_j^{(k)}-x_j\|=O(q_j^k) \tag{6}$$

这里的 q_j 同式（5）。

上述的同时迭代的各种算法，在 E_k 上计算具有上述性质的 $x_j^{(k)}$ 时，使用的方法各不相同。本文给出的算法，将采用在 E_k 上计算系统

$$(KM^{-1}K)x=\lambda^2Mx \tag{7}$$

的特征方向 $X_k=(x_1^{(k)},\cdots\cdots,x_p^{(k)})$，它满足于

$$X_k^TMX_k=I_p \tag{8}$$

$$X_k^T(KM^{-1}K)X_k=D_k^2 \tag{9}$$

其中 D_k 为 p 阶对角矩阵，并自始至终假定它的对角线元素 $d_{jj}^{(k)}$ 满足于

$$d_{11}^{(k)}\leqslant d_{22}^{(k)}\leqslant\cdots\cdots\leqslant d_{pp}^{(k)} \tag{10}$$

在 E_k 上计算上述 X_k 并不困难（见下节），而且这样的 X_k 还具有下面的一个极为重要的性质。

定理：在稳定收敛情况下，如果 X_k 为系统（7）在 E_k 上的特征方向，则有

$$\|x_j^{(k)}-x_j\|=O(q_j^{(k)}) \tag{11}$$

$$|d_j^{(k)}-\lambda_j|=O(q_j^{2k}) \tag{12}$$

其中 q_j 同式（5）。证明请参见文献〔1〕。

三、算　法

在 n 维空间中，任取一初始 $n\times p$ 矩阵 $S_0=(s_1^{(0)},\cdots\cdots,s_p^{(0)})$，然后用下列迭代格式逐步生成新的 S_k：

1　解方程组，求出 Z_k：

$$KZ_k=S_{k-1}$$

2　Gram-Schmidt 直交化，求出 Y_k 和 R_k：

$$MZ_k=Y_kR_k$$

$$(Y_k^TM^{-1}Y_k=I_p,\ R_k\text{ 为上三角})$$

3　形成投影矩阵 B_k：

$$B_k=R_kR_k^T$$

4　解投影矩阵 B_k 的特征值问题：

$$Q_k^TB_kQ_k=D_k^2\ (D_k\text{ 为对角阵})$$

5　形成下步所需的 S_k：

$$S_k=Y_kQ_k \tag{13}$$

如果在上述算法中，令

$$S_k=MX_k \tag{14}$$

则不难证明 X_k 为系统（7）在 E_k 子空间上的特征方向。事实上，

$$X_k^TMX_k=Q_k^TY_k^TM^{-1}Y_kQ_k=I_p \tag{15}$$

$$X_k^T(KM^{-1}K)X_k=Q_k^T(R_kR_k^T)^{-1}Q_k=D_k^2 \tag{16}$$

所以，由上节定理可知，算法（13）的收敛率为（11），（12）。事实上，（11），（12）式是上述同时迭代各种算法所共有的收敛率。

下面先叙述算法（13）在计算机上实现方面的几个值得注意的问题，然后用一个框图（见图1）描述全部计算过程。

1　初始 p 个试向量的选取可采用随机化过程〔2〕，或采用 Bathe 的选取办法〔8〕。

2　迭代格式 2 是使用 Gram-Schmidt 直交化过程实现的。事实上，若令

$$Z_k=X_kR_k\ (\text{Gram-Schmidt 直交化}) \tag{17}$$

其中 $X_k^TMX_k=I_p$，R_k 为上三角矩阵。在上式两边同时左乘以 M，并令 $Y_k=MX_k$，则不难证明

$$Y_k^TM^{-1}Y_k=I_p \tag{18}$$

有关 Y_k 的计算参见图 1。

3　由于算法（13）可采用降维处理技巧，所以当某些试向量一旦满足了收敛条件，将不再让它们参加以后的迭代。但由于计算误差的原因，在以后的迭代中，还需在其余的试向量上“净化”掉那些已收敛的向量。另外，还需注意，算法（13）中的 B_k 的阶数将随着试向量已收敛的个数的增加而不断减小。图 1 中的上三角矩阵 R_g 是 p 阶上三角矩阵 $R=(r_{ij})$ 的右下角 $p-q$ 阶矩阵。其中 q 为已收敛的特征对个数。

4　当迭代 l 步之后，待求的 q 个特征对全部收敛时，$\boldsymbol{D}_l$ 中的对角线元素 $d_{jj}^{(l)}$ 就是特征值 λ_j 的近似，而 $\boldsymbol{S}_1$ 的列 $s_j^{(l)}$（$j=1, 2, \cdots\cdots, q$）还不是特征向量 x_j 的近似向量 $x_j^{(l)}$，但不难看出，二者有下面关系：

$$s_j^{(l)}=\boldsymbol{M}x_j^{(l)} \quad (j=1, \cdots\cdots, q) \tag{19}$$

为了获得 $x_j^{(l)}$，还需做下面少量工作：

$$\boldsymbol{K}z_j^{(l)}=s_j^{(l)} \tag{20}$$

$$x_j^{(l)}=d_{jj}^{(l)}z_j^{(l)} \quad (j=1, \cdots\cdots, q) \tag{21}$$

一般地，试向量个数 p 由下式确定：

$$p=\min\{q+8, 2q\} \tag{22}$$

5　收敛准则可参阅文献〔2, 8〕，建议使用文献〔8〕中的准则：

$$\left|\frac{d_{jj}^{(k)}-d_{jj}^{(k-1)}}{d_{jj}^{(k-1)}}\right|<\varepsilon^2 \tag{23}$$

其中 ε 为特征向量的允许误差。

6　框图中的 $\boldsymbol{X}$, $\boldsymbol{Y}$, $\boldsymbol{Z}$, $\boldsymbol{S}$ 可以使用同一个储存位置。

7　由于同时迭代的各种算法都可以使用切比雪夫多项式或原点移位法进行加速，所以在框图中不再叙述。有兴趣的读者可以参阅文献〔1、2、6〕。

8　由于 $\boldsymbol{M}$ 阵在迭代过程中始终保持不变，所以建议使用排除全部零元素的压缩存储办法。

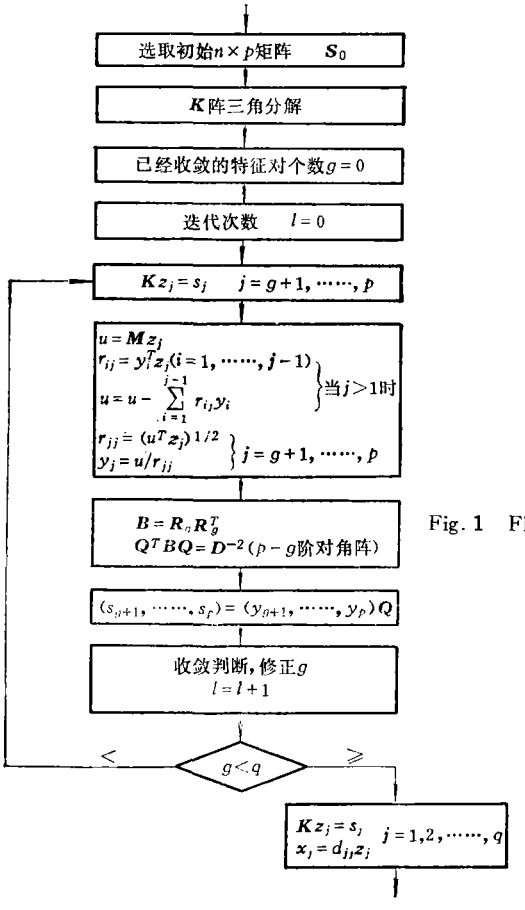

图 1　本文算法的计算流程

Fig. 1　Flow diagram for the proposed algorithm

表1 几种算法的比较

Table 1 Comparison of several algorithms

算法	Rutishauer[1]	Reinsch[2]	McCormick[9]	Nicolai[10]	Bathe[8]	本文
迭代格式	$AZ_k = Y_{k-1}$ $B_k = Z_k^T Z_k$ $Q_k^T B_k Q_k = D_k^{-2}$ $Y_k = Z_k Q_k D_k$	$A\bar{Z}_k = Y_{k-1}$ $\bar{Z}_k = Z_k R_k$ ($Z_k^T Z_k = IR_k$ 为上三角) $B_k = R_k R_k^T$ $Q_k B_k Q_k = D_k^{-2}$ $Y_k = Z_k Q_k$	$KZ_k = MX_{k-1}$ $B_k = Z_k^T M Z_k$ $Q_k^T B_k Q_k = D_k^{-2}$ $X_k = Z_k Q_k D_k$	$K\bar{Z}_k = MX_{k-1}$ $\bar{Z}_k = Z_k R_k$ ($Z_k^T M Z_k = IR_k$ 为上三角) $B_k = R_k R_k^T$ $Q_k B_k Q_k = D_k^{-2}$ $X_k = Z_k Q_k$	$KZ_k = X_{k-1}$ $Y_k = MZ_k$ $K_k = X_{k-1}^T Z_k$ $M_k = Z_k^T Y_k$ $K_k Q_k = M_k Q_k D_k$ $X_k = Y_k Q_k$	$KZ_k = S_{k-1}$ $MZ_k = Y_k R_k$ ($Y_k^T M^{-1} Y_k = IR_k$ 为上三角) $B_k = R_k R_k^T$ $Q_k^T B_k Q_k = D_k^{-2}$ $S_k = Y_k Q_k$
M分解否	分解 $M = LL^T$ $A = L^{-1} K L^{-T}$ $Y = L^T X$	分解 $M = LL^T$ $A = L^{-1} K L^{-T}$ $Y = L^T X$	不分解	不分解	不分解	不分解
是否可降维处理	不能	能	不能	能	不能	能
投影矩阵形式	标准	标准	标准	标准	广义	标准
每次迭代的运算量	$2np(b_k + b_m)$ $+\frac{3}{2} np^2 + J(p)$	$np(b_k + b_m) + np^2$ $+\left(\sum_{i=1}^{p} J(i)\right)/p$	$2np(b_k + b_m)$ $+ 2npb_m + np^2$ $+\left(\sum_{i=1}^{p} J(i)\right)/p$	$np(b_k + b_m)$ $+ npb_m + np^2$ $+\left(\sum_{i=1}^{p} J(i)\right)/p$	$2np(b_k + b_m)$ $+ 2np^2 + 2J(p)$	$np(b_k + b_m) + np^2$ $+\left(\sum_{i=1}^{p} J(i)\right)/p$

四、方法比较

这一节，我们将比较本文算法与其它常用的几种算法。在比较之前，先做几点说明：

1 设 b_k 和 b_m 分别为 $\boldsymbol{K}$ 和 $\boldsymbol{M}$ 的半带宽。虽然在 $\boldsymbol{M}$ 不分解的算法中，$\boldsymbol{M}$ 可以按压缩方案储存，但为了方便起见，仍用 b_m 来统计运算量。

2 在可以采用“降维”处理的算法中，在统计每步运算量时，使用了平均值的方法。如果设已有 i 个特征对收敛，这时每步迭代所需的运算量为 $T(p-i)$，那么平均每步运算量为：

$$\frac{\sum_{i=0}^{p-1} T(p-i)}{p} \tag{24}$$

3 用 $J(p)$ 代表解 p 阶对称矩阵全部特征值问题所用的运算量。用 $2J(p)$ 近似表示解相应广义特征值问题的运算量。

几种算法的比较情况列入表 1。

从表 1 显然可得出下列结论：

本文给出的算法

1 比 Rutishauer，Reinsch 算法少做一次 $\boldsymbol{M}$ 阵的分解。而且，当 $\boldsymbol{M}$ 按压缩方案储存时，每步迭代的运算量实际上要比 Reinsch 的运算量小。

2 每步迭代中，比 McCormick 和 Nicolai 算法少做一次 $\boldsymbol{M}$ 阵与试向量的乘积。

3 与 Bathe 算法相比，除了可进行降维处理外，毋须每步迭代解一个 p 阶广义特征值问题。

五、数值检验

为了检验本文算法的可靠性和效率，在 SIEMENS-7760 计算机上使用FORTRAN 语言做了多方面的数值检验工作。现把有关结果列出如下。

1 算例 1。使用 Reinsch 方法和本文方法，计算某飞机全机特征值问题。本算例主要是用来比较两种方法的收敛情况（见表 2）和结果精度（见表 3）。本算例使用的试向量为 25 个，待求特征对17个。所求出的17个特征值仅在小数点后10位上有差别，限于篇幅，不再列表。

表 2 Reinsch 算法和本文算法的收敛情况

Table 2 Comparison of Reinsch's algorithm with the proposed one in convergence

迭代次数		1	2	3	4	5	6	7	8	9	10
收敛个数	本文	0	0	0	0	4	14	15	17		
	Reinsch	0	0	0	2	11	15	16	16	17	

表 3　Reinsch 算法和本文算法的特征向量精度

Table 3　Comparison of Reinsch's algorithm with the proposed one in eigenvector accuracy

特征向量序号	1	2	3	4	5	6	7	8
本文(10^{-4})	0.0064	0.0143	0.0559	0.0062	0.0830	0.0403	0.0064	0.0007
Reinsch(10^{-4})	0.0100	0.1231	0.0653	0.0040	0.0188	0.0251	0.0219	0.0447

2　算例 2。阶数为 500，K 和 M 的半带宽为 301。本算例主要用来考核各算法的计算效率。采用 13 个试向量，计算 5 个特征对（见表 4）。

表 4　几种算法的计算时间

Table 4　Computer run time of several algorithms

算　法	Rutishauer	Reinsch	McCormick	Nicolai	Bathe	本　文
时间(秒)	1106	1069	992	973	780	758

3　算例 3。分别用 Bathe 算法和本文算法计算了三个特征值问题。本算例为了证实当待求特征对比较多时本文算法要比 Bathe 算法效率高（见表 6）。

表 5　三个问题的参数

Table 5　Parameters for three problems

问　题	阶　数	半　带　宽	试向量个数	待求特征对数
1	100	5	30	25
2	200	20	33	25
3	500	10	38	30

表 6　Bathe和本文算法的计算时间

Table 6　Comparison of Bathe's algorithm with the proposed one in computer run time

问　题	1	2	3
本　文（秒）	93	234	558
Bathe （秒）	194	363	986

六、结　论

本文提出了同时迭代的一种变型算法，证明了它的收敛率，讨论了在计算机上实现的问题，并与常用的几种同时迭代算法做了比较。可以断定，系统（1）的规模和带宽越大，本文算法与 Rutishauer，Reinsch，McCormick 和 Nicolai 算法相比计算效率就越高；而系

统（1）的待求特征对越多，本文算法与 Bathe 算法相比计算效率就越高。

本方法在研究期间，得到了管德同志的具体指导和支持，并得到李光权同志的帮助，在此深表感谢。

参 考 文 献

[1] H. Rutishauer, Computational aspects of F. L. Bauer's simultaneous iteration method, Numer Math 13, 4-11, 1969.

[2] H. Rutishauer, Simutaneous iteration method for symmetric matrices, Numer Math 16, 205-223, 1970.

[3] A. Jennings, A direct iteration method for obtaining the latent roots and vectors of symmetric matrix, Proc. Comb. Phil. Soc., 63, 755-765, 1967.

[4] M. Clint and A. Jennings, The evaluation of eigenvalues and eigenvectors of real symmetric matrices by simultaneous iteration, Computer J., 13, 76-80, 1970.

[5] R. B. Corr and A. Jennings, A Simultaneous iteration algorithm for symmetric eigenvalue problems, Int. J. Num Meth. Engng, 10, 647-663, 1976.

[6] Yamamoto and Ohtsubo, Subspace iteration accelerated by using Chebyshev polynomials for eigenvalue problems with symmetric matrices, Int. J. Num Meth. Engng, 10, 935-944, 1976.

[7] K. J. Bathe and E. L. Wilson, Large eigenvalue problems in dynamic analysis, J. Engng Meth. Div., Proc. ASCE EM6, 1471-1485, 1972.

[8] K. J. Bathe and E. L. Wilson, Numerical Methods in Finite Element Analysis, Englewood Cliffs, 1976.

[9] S. F. McCormick and T. Noe, Simultaneous iteration for the matrix eigenvalue problem, J. Linear Algebra & its Application, 16, 43-56, 1977.

[10] P. J. Nicolai, Eigenvectors and eigenvalues of real generalized symmetric matrices by simultaneous iteration, ALGORITHM 538 Collected Algorithms from ACM, 1979.

A NEW SIMULTANEOUS ITERATION ALGORITHM FOR EVALUATION OF EIGENPROBLEMS IN LARGE STRUCTURES

Liu Guoguang and Li Junjie

(*Aircraft Structural Mechanics Research Institute*)

Abstract

A new improved algorithm of simultaneous iteration is presented for evaluation of generalized eigenproblems in dynamic analysis of large structures. The convergence of this algorithm is proved by the concepts of E_K subspace and eigendirection and some details of how to perform this algorithm in computer are discussed.

Based on statistics of computational quantity required for each step of iteration and a great number of practical computations, it is shown that in convergence rate the present method is comparable to some simultaneous or subspace iteration algorithms available up to now, but it can cut down the computer run time for each step of iteration and raise the computation efficiency.