대형 유한요소 고유치 해석에서의 부공간 축차법 효율 개선

주 병 현† · 이 병 채*

(2002년 8월 10일 접수, 2003년 2월 8일 심사완료)

Improvement of Computational Efficiency of the Subspace Iteration Method for Large Finite Element Models

Byung-Hyun Joo and Byung-Chai Lee

Key Words : Subspace Iteration Method(부공간 축차법), Large Finite Element Model(대형 유한요소 모델), Eigenvalue Problem(고유치 문제)

Abstract

An efficient and reliable subspace iteration algorithm using the block algorithm is proposed. The block algorithm is the method dividing eigenpairs into several blocks when a lot of eigenpairs are required. One of the key for the faster convergence is carefully selected initial vectors. As the initial vectors, the proposed method uses the modified Ritz vectors for guaranteering all the required eigenpairs and the quasi-static Ritz vectors for accelerating convergency of high frequency eigenvectors. Applying the quasi-static Ritz vectors, a shift is always required, and the proper shift based on the geometric average is proposed. To maximize efficiency, this paper estimates the proper number of blocks based on the theoretical amount of calculation in the subspace iteration. And it also considers the problems generated in the process of combining various algorithms and the solutions to the problems. Several numerical experiments show that the proposed subspace iteration algorithm is very efficient, reliable ,and accurate.

1. 서 론

많은 공학문제에서 시스템의 고유치와 고유벡터를 구하는 고유치 문제를 접하게 된다. 그러나 고유치 해석을 하는 것은 정적 해석을 하는 경우보다 계산량이 매우 많기 때문에 많은 연구자들이 고유치를 구하는 효율적인 알고리즘을 개발하기 위해 노력해 왔다.[1]

컴퓨터의 발전에 따라 모델의 복잡도도 크게 증가하고 있기 때문에 몇 개의 관심있는 고유치만을 구하는 방법이 더욱 실용적이게 되었고 이런 방법들 중 대표적인 방법으로 부공간 축차법이 있다. 부공간 축차법은 동시 역축차 방법과 레일리 리츠 해석(Rayleigh-Ritz analysis)을 결합한 방법이다. 이 방법은 강건성이나 수치적인 안정성은 뛰어나지만, 적절한 초기벡터가 선택되지 않을 경우 고유치를 빠뜨리거나 축차회수가 많아지게 되고, 고유치가 거의 같거나 가깝게 모여있는 경우 고유치의 수렴이 매우 느려질 수 있으며, 구하고자 하는 고유치의 갯수가 많을 경우 부공간 고유치 문제를 푸는 데도 계산상의 노력이 많이 들고, 많은 주메모리를 사용하게 된다는 단점이 있다.[2-4]

이런 문제점들을 해결하기 위하여 다양한 방법들이 제안되었다. Jennings와 Stewart[5]는 동시 축차 방법에서 이미 수렴한 벡터의 재계산을 하지 않는 고정(locking) 알고리즘을 제안하였고, Akl[6]은 선형 방정식을 축차로 풀 때 축차회수를 줄일 수 있는 가속완화법(overrelaxation)을 부공간 축차법에 적용하였다. Bathe와 Ramaswamy[7]는 가속완화법의 적

* 한국과학기술원 기계공학과
† 책임저자, 회원, 한국과학기술원 기계공학과
E-mail : bclee@mail.kaist.ac.kr
TEL : (042)869-3031 FAX : (042) 869-3095

용과 적절한 이동(shift)값의 선택 그리고 이동의 효율성에 대해 이론적인 기초를 확립하였다. 또한 구하고자 하는 고유치의 갯수가 많을 경우 고유치를 여러 개의 블록으로 나누어 구하는 블록 알고리즘에 대해 언급하였다. Wilson과 Itoh[4]는 부공간 축차법의 문제점에 대해 언급하고 이 문제점의 해결에 대한 개선 방안을 제시하였다. 또한 블록 알고리즘과 이동을 적절히 조합하여 블록에서 다음 블록으로 넘어갈 때 적절히 이동을 하여 고유치를 구하는 새로운 블록 알고리즘을 제안하였다.

부공간 축차의 가속화를 위하여 다양한 방법들이 제안되었지만, 어떤 문제에도 효율적으로 적용될 수 있는 방법은 다음과 같다.

(1) 적절한 초기벡터를 선택하여 초기벡터에 의해 만들어지는 부공간이 구하고자 하는 고유벡터에 의해 만들어지는 부공간과 가까우면서도 원하는 고유치를 빠뜨리지 않도록 한다.

(2) 고정 알고리즘 을 적용하여 이미 수렴한 벡터의 재계산을 방지한다.

(3) 적절한 이동값을 선택하여 축차회수를 가급적 많이 줄인다.

(4) 블록 알고리즘을 적용하여 사용되는 주메모리의 양을 줄인다.

본 논문에서는 적절한 초기벡터의 선택, 고정 알고리즘과 이동값의 적용, 주메모리의 사용량을 줄일 수 있는 블록 알고리즘을 적절히 결합하여 대형 문제나 고유치를 많이 구하는 문제뿐만 아니라 작은 문제에도 효율적으로 적용할 수 있는 알고리즘을 제안하려고 한다. 또한 여러 알고리즘들이 결합되는 과정에서 생길 수 있는 문제점들을 파악하고 적절한 해결책을 제시한다.

2. 부공간 축차의 가속화 방법

2.1 기본적인 부공간 축차 알고리즘

부공간 축차법은 자유도가 n 인 시스템에서 다음의 식을 만족하는 가장 작은 p 개의 고유치, λ_i 와 이에 대응하는 고유벡터, $\boldsymbol{\varphi}_i$ 를 구하기 위한 방법이다.

$$\mathbf{K}\boldsymbol{\Phi} = \mathbf{M}\boldsymbol{\Phi}\boldsymbol{\Lambda}$$
$$where\ \boldsymbol{\Lambda} = diag(\lambda_i)\ and\ \boldsymbol{\Phi} = [\boldsymbol{\varphi}_1, \ldots, \boldsymbol{\varphi}_p] \quad (1)$$

부공간 축차법의 기본 원리는 동시 역축차 방법과 레일리 리츠 해석을 결합한 방법으로 기본적인 알고리즘은 Table 1과 같다.[1] 만약 $\mathbf{X}_1$ 이 구하고자 하는 고유벡터에 직교하지 않을 경우 다음과 같은 결과를 얻을 수 있다.

Table 1 Algorithm of the subspace iteration method

1. Set q M-orthonormalized initial vectors $\mathbf{X}_1$
2. For $k = 1, 2, \ldots$; begin the iteration:
   - 2.1 $\mathbf{Y}_k = \mathbf{M}\mathbf{X}_k$: $0.5nm^2 + 1.5nm$
   - 2.2 $\mathbf{K}\bar{\mathbf{X}}_{k+1} = \mathbf{Y}_k$: $nq(2m+1)$
   - 2.3 $\mathbf{K}_{k+1} = \bar{\mathbf{X}}_{k+1}^T \mathbf{Y}_k$: $0.5nq(q+1)$
   - 2.4 $\bar{\mathbf{Y}}_{k+1} = \mathbf{M}\bar{\mathbf{X}}_{k+1}$: nq
   - 2.5 $\mathbf{M}_{k+1} = \bar{\mathbf{X}}_{k+1}^T \bar{\mathbf{Y}}_{k+1}$: $0.5nq(q+1)$
   - 2.6 Solve the reduced eigenproblem: $\mathbf{K}_{k+1}\mathbf{Q}_{k+1} = \mathbf{M}_{k+1}\mathbf{Q}_{k+1}\boldsymbol{\Lambda}_{k+1}$: $o(q^3)$ neglected
   - 2.7 $\mathbf{Y}_{k+1} = \bar{\mathbf{Y}}_{k+1}\mathbf{Q}_{k+1}$: nq^2
   - 2.8 If first p eigenpairs are converged, exit the loop;
   - 2.9 End the loop;
3. Stop the calculation.

$$\boldsymbol{\Lambda}_{k+1} \to \boldsymbol{\Lambda}\ and\ \mathbf{X}_{k+1} \to \boldsymbol{\Phi}\ as\ k \to \infty \quad (2)$$

Table 1에서 오른쪽에 있는 식은 이론적인 계산량을 대략적으로 나타낸 것으로 n 은 시스템의 자유도, m 은 평균 밴드폭의 반, q 는 부공간의 크기이다.

2.2 부공간 축차의 대표적인 가속화 방법

2.2.1 재계산을 배제한 방법(locking)

기본적인 부공간 축차법에서 이미 수렴한 축차벡터들은 다음 축차에서 계산을 하더라도 값이 거의 변하지 않음에도 불구하고 매번 축차를 할 때마다 계산을 반복해서 하였다. 만약 수렴한 축차벡터를 다음 축차부터 계산에서 제외시킨다면 계산효율은 증가된다.

2.2.2 이동 방법

이동이란 부공간 축차과정에서 수렴이 진행될수록 수렴속도가 느려지는 현상을 방지하기 위해 강성행렬을 다음과 같이 바꾸는 것을 의미한다.

$$\mathbf{K}' = \mathbf{K} - \mu\mathbf{M} \quad (3)$$

이동이 된 후 고유치의 이론적인 수렴률은 다음과 같이 바뀌게 된다.

$$\frac{\lambda_i}{\lambda_{q+1}} \rightarrow \left|\frac{\lambda_i-\mu}{\lambda_{q+1}-\mu}\right| \tag{4}$$

Bathe와 Ramaswamy[7]는 이동 알고리즘을 체계적으로 부공간 축차에 적용하였다. 또한 이동을 실제 부공간 축차법에 적용할 때 발생할 수 있는 문제점과 이에 대한 해결책을 제시하였고, 이동의 효율성에 대해서 언급하였다.

2.2.3 블록 알고리즘

일반적으로 구하고자 하는 고유치의 갯수가 많을 경우($p>50$) 부공간 축차법의 효율은 급격히 떨어지게 된다. 수렴속도가 느려질 뿐만 아니라 요구되는 메모리의 양도 증가하게 되어 주메모리에서 처리하지 못하는 경우 컴퓨터는 임의적으로 보조 기억 장치의 일부를 가상메모리로 잡아서 주메모리처럼 사용한다. 이런 경우 계산시간이 급격히 증가할 수 있다. 블록 알고리즘을 사용할 경우 보조 기억 장치로의 접근을 보다 더 계획적으로 할 수 있게 되며 결과적으로 많은 시간절약을 할 수 있게 된다.

Bathe와 Ramaswamy[7]는 구하고자 하는 고유치의 갯수가 많을 경우 여러 개의 블록으로 나누어 계산하는 것이 효율적이라고 하였다. 또한 이동 알고리즘을 적용하여 각 블록마다 축차를 진행하다가 이동이 효율적인지 판단하여 이동을 하였다. 이에 반하여 Wilson과 Itoh[4]은 블록 알고리즘을 적용하되 다음 블록으로 넘어갈 때 항상 이동을 해주는 방식의 알고리즘을 제안하였다.

3. 부공간 축차법의 성능 개선

3.1 초기벡터의 개선

3.1.1 수정된 리츠벡터의 적용

적절하게 선택된 리츠벡터의 집합은 고유치 해석에서 초기벡터로 사용될 경우 축차과정을 상당히 가속화시킬 수 있는 것으로 알려져 있다.[8]

보통의 리츠벡터는 하나의 초기벡터로부터 정적 재귀(static recurrence)과정과 M-직교화를 통해 만들어진다. 하나의 초기벡터로부터 만들어지기 때문에 이 하나의 초기벡터에 의해 구하고자 하는 모든 고유벡터가 가진될 수 있어야 한다. 그러나 현실적으로 그렇게 되기는 상당히 힘들고 중근이 많은 문제인 경우 고유치를 놓치는 경우가 많이 발생할 수도 있다.[9]

그러므로 여러 개의 초기벡터를 사용하여 리츠벡터를 만든다면 구하고자 하는 고유벡터를 가진시킬 확률이 그만큼 높아지게 된다. 수정된 리츠벡터에 대한 알고리즘은 Table 2와 같다. 이때 $\mathbf{x}_{0i}$는 Bathe가 제안한 초기벡터를 사용하며, 갯수는 이론적으로 구하고자 하는 고유치 중 중근의 갯수보다 많게 정하면 된다. 보통 $2<t<p$이면 적합하다고 알려져 있다.

Table 2 Algorithm for generation of orthogonal modified Ritz vectors

1. For $i=1,\ldots,t$ choose initial vectors $\mathbf{x}_{0i}$.
2. Calculate m Ritz vectors.

Let $\mathbf{q}_0=0$, for $k=1,2,\ldots,m$, and calculate

2.1 $\mathbf{K}\bar{\mathbf{q}}_k^{(1)}=\mathbf{M}\tilde{\mathbf{q}}_{k-1}$, where

$$\tilde{\mathbf{q}}_{k-1}=\begin{cases}\mathbf{q}_{k-1}+\mathbf{x}_{0k} & k=1,2,\ldots,t\\ \mathbf{q}_{k-1} & k=t+1,t+2,\ldots,m\end{cases}$$

2.2 Orthogonalize.

$\bar{\mathbf{q}}_k^{(s+1)}=\bar{\mathbf{q}}_k^{(s)}-\sum_{i=1}^{k-1}\alpha_i^{(s)}\mathbf{q}_i$, where

$\alpha_i^{(s)}=\bar{\mathbf{q}}_k^{(s)}\mathbf{M}\mathbf{q}_i \quad s=1,2,\ldots \quad i=1,2,\ldots,k-1$

2.3 Normalize.

$\mathbf{q}_k=\bar{\mathbf{q}}_k^{(s+1)}/\beta_k$, where

$\beta_k=(\bar{\mathbf{q}}_k^{(s+1)T}\mathbf{M}\bar{\mathbf{q}}_k^{(s+1)})^{1/2}$

Table 3 Comparison of the classical and modified Gram-Schmidt algorithm

Classical Gram-Schmidt	Modified Gram-Schmidt
for $j=1$ to n	for $i=1$ to n
$\mathbf{v}_j=\mathbf{a}_j$	$\mathbf{v}_i=\mathbf{a}_i$
for $i=1$ *to* $j-1$	for $i=1$ to n
$r_{ij}=\mathbf{q}_i^T\mathbf{a}_j$	$r_{ii}=(\mathbf{v}_i^T\mathbf{v}_i)^{1/2}$
$\mathbf{v}_j=\mathbf{v}_j-r_{ij}\mathbf{q}_i$	$\mathbf{q}_i=\dfrac{\mathbf{v}_i}{r_{ii}}$
$r_{jj}=(\mathbf{v}_j^T\mathbf{v}_j)^{1/2}$	**for** $j=i+1$ **to** n
$\mathbf{q}_j=\dfrac{\mathbf{v}_j}{r_{jj}}$	$r_{ij}=\mathbf{q}_i^T\mathbf{v}_j$
	$\mathbf{v}_j=\mathbf{v}_j-r_{ij}\mathbf{q}_i$

3.1.2 직교화의 유지

리츠벡터를 만드는 과정은 란초스 벡터를 만드는 과정과 상당히 유사하므로 직교성이 훼손될 수 있다. 그러나 란초스 벡터는 이전의 두 란초스 벡터에 대해서 직교화를 하는 반면 리츠벡터는 이전에 만들어진 모든 리츠벡터에 대해서 직교화를 하기 때문에 직교화가 훼손되는 정도가 덜하다. 그러므로 비교적 간단한 알고리즘 변화를 통해 직교성을 강화할 수 있다. Table 3은 전형적인 Gram-Schmidt 직교화와 수정된 Gram-Schmidt 직교화 과정을 비교한 것이다. 알고리즘은 다르지만 수학적으로 동일하다.[10]

3.2 블록 알고리즘의 적용

일반적으로 리츠벡터는 구하고자 하는 고유치의 2배로 축차벡터의 갯수를 설정하기 때문에 구하고자 하는 고유치의 갯수가 많을 경우 Bathe가 제안한 초기벡터의 갯수인 $q=\min(p+8,2p)$ 보다 상당히 커지기 때문에 주메모리에서 풀지 못하는 경우가 더 많을 것이다. 그러므로 리츠벡터를 부공간 축차법의 초기벡터로 적용하려면 블록 알고리즘이 더 필요하게 된다.

3.2.1 준정적 리츠벡터의 적용[8]

표준 리츠벡터를 블록 알고리즘에 적용할 때 뒤쪽의 블록으로 갈수록 리츠벡터의 성능이 떨어지는 현상이 나타난다. 이것은 표준 리츠벡터가 진동수가 0인 상태에서의 정적 재귀에 의해 만들어지기 때문이다. 이런 이유로 낮은 고유치에 대응하는 고유벡터를 표현하기에 적합하지만 만약 높은 고유치, 즉 높은 진동수에 대응하는 고유벡터를 표현하려면 훨씬 더 많은 리츠벡터가 필요하게 되어 부공간 축차법의 효율이 떨어지게 된다.

그러나 거의 같은 수의 리츠벡터를 사용하더라도 원하는 진동수 근처에서의 고유벡터를 효율적으로 표현할 수 있는 방법을 사용하면 이를 완화할 수 있다. 이것은 리츠벡터를 만들 때 강성행렬을 원하는 고유치 근처로 이동시켜 만드는 것이다. 즉 이전의 리츠벡터를 만드는 방법에서 $\mathbf{K}$ 를 $\mathbf{K}-\mu\mathbf{M}$ 으로 바꾸면 된다. 이렇게 해서 만들어진 리츠벡터는 고유치 μ 근처의 고유벡터를 효율적으로 표현할 수 있다.

3.2.2 블록 알고리즘의 변형

Wilson과 Itoh[4]는 여러 개의 이동값 근처에서 고유치를 그룹단위로 계산해내는 알고리즘을 제안하였다. 이 알고리즘은 한 블록에서 다음 블록으로 넘어갈 때 항상 이동을 해주는 방식이다. 준정적 리츠벡터를 적용할 경우 이런 방식으로 블록 알고리즘이 바뀌어야 한다. 이것은 다음 블록에서 초기벡터를 만들 때 이동이 필요하기 때문이다.

3.3 개선된 블록 알고리즘

3.3.1 기본 알고리즘

블록 알고리즘의 근본적인 목적은 주메모리에서 문제를 풀 수 없을 경우 보조 기억 장치로의 효율적인 접근을 통하여 주메모리의 사용을 극대화하는 것이 목적이다. 그러나 여기에서 개선된 블록 알고리즘의 목적은 주메모리에서 풀 수 있는 문제라 할지라도 적절하게 블록을 나누어 풀 경우 주메모리를 적게 사용하면서도 기존의 방법보다 훨씬 더 시간을 절약할 수 있도록 하는 것이며 동시에 구하고자 하는 고유치를 빠뜨리지 않는 것이다. 대략적인 알고리즘은 다음과 같다.

(1) 초기벡터로 준정적 리츠벡터와 수정된 리츠벡터를 결합한 방법을 사용하며, 리츠벡터를 만드는 과정에서 직교화를 할 경우 수정된 Gram-Schmidt직교화를 이용한다. 이렇게 하여 $2p$ 개의 초기 축차벡터를 만든다. 단 초기 이동은 0이다.

(2) 초기 축차벡터를 만든 후 부공간 축차를 하여 원하는 p 개의 고유치와 대응하는 고유벡터를 구한 후 고유벡터는 보조 기억 장치에 저장한다.

(3) 이 블록에서 남은 p 개의 고유치들을 이용하여 다음 블록에서 사용될 이동값을 구한 후 다시 (1)로 간다.

대략적인 알고리즘은 간단하지만 고려해야 할 중요한 사항들이 있다.

3.3.2 이전 블록에서 수렴하지 못한 축차벡터의 이용

한 블록에서 p 개의 고유치를 구하고자 한다면 $2p$ 개의 축차벡터를 사용하므로 p 개는 원하는 고유벡터로 수렴하게 되고 p 개가 남게 된다. 그러나 이 축차벡터들도 상당히 계산이 진행된 상태이고 이미 수렴한 상태에 있는 것도 있기 때문에 다음 블록에서 이용하는 것이 좋을 것이다.

이런 이유로 이전 블록에서 넘어온 p 개의 축차벡터를 수정된 리츠벡터를 만들 때 수정된 리츠벡터의 초기벡터인 $\mathbf{x}_{0_i}$ 에 대입하는 방법을 생각하였다. 이렇게 할 경우 이미 이전 블록에서 어느 정도 수렴한 벡터는 그 값이 그대로 나오게 되고

리츠벡터를 만드는 과정에서 $\mathbf{K}^{-1}\mathbf{M}$ 이 행해지므로 수렴에 가깝지 않던 벡터들도 좀더 고유벡터에 더 가깝게 되는 효과가 있다.

3.3.3 적절한 이동값의 선택

만약 구하고자 하는 고유치의 범위를 정확하게 알고 있다면 이동값은 고유치 스펙트럼의 중앙값으로 택하는 것이 가장 이상적이다.[4] 현재의 알고리즘에서는 다음 블록의 고유치가 정확히 어느 범위에 있는지 모르기 때문에 적절한 값을 예측해야 한다.

여기에서 제안된 이동값은 이전 블록에서 남은 p 개의 고유치를 이용하여 구하게 되는데 기하평균의 개념과 적절하게 합치된다. 기하평균은 변량의 극단적인 값의 변화에 영향을 덜 받는 특징이 있다. 일반적으로 수렴하지 못한 고유치들은 상계(upper bound) 접근을 하기 때문에 실제의 고유치보다 크다. 수렴정도에 따라 차이가 있지만 실제 고유치보다 10의 몇 승 정도로 큰 값들이 있는 경우도 있다. 그러므로 다음과 같이 하였다.

이전 블록에서 남은 p 개의 고유치에 대한 예상값을 $p/2$ 개씩 순서대로 2개의 집합으로 나눈다. 만약 p 가 홀수인 경우 두번째 집합의 갯수가 1개 더 많게 한다. 첫번째 집합에서 허용공차인 10^{-6} 으로 수렴한 고유치를 p^* 개라 하면 이 고유치들에 기하평균을 취한 값을 α 라 한다. 그리고 두번째 집합에서는 크기가 작은 순서로 p^* 개의 값을 선택하여 기하평균을 취한 값을 β 라 한다. 그러면 이동값은 다음과 같다.

$$\mu = \frac{\alpha+\beta}{2} \tag{5}$$

만약 10^{-6} 으로 수렴한 값이 없을 경우 첫번째 집합의 가장 작은 값을 α 라 하고 마찬가지로 두번째 집합에서 가장 작은 값을 β 라 한다. 이렇게 이동값을 정한 것은 이론적으로 고유치가 작을수록 더 빨리 수렴해야 하지만 실제 그렇지 않기 때문에 이론에 근거하여 이동값을 결정하는 것에는 한계가 있기 때문이다. 그러므로 어느 정도 수긍할 수 있는 대략적인 값들을 바탕으로 구하고자 하는 고유치 스펙트럼의 중앙 근처에 이동값이 올 수 있도록 하였다.

3.3.4 적절한 블록갯수 정하기

여기에서 사용되는 블록 알고리즘은 한 블록에서 다음 블록으로 넘어갈 때 항상 이동을 해야 한다. 이것은 매 블록마다 한 번의 강성행렬 분해를 해야 한다는 것을 의미한다. 작은 문제를 풀 경우 크게 상관없지만 대형 문제를 풀 경우 강성행렬의 분해시간이 상당히 크기 때문에 블록을 너무 많이 나누는 것은 오히려 시간의 낭비를 가져올 수 있다. 이것은 블록 알고리즘의 효율을 극대화하기 위해 적절한 블록의 갯수를 정해야 한다는 것을 뜻한다. 그러므로 불합리하게 블록의 갯수를 정하는 것보다 어느 정도 계산량에 근거하여 블록의 갯수를 정하는 것이 바람직하다. 블록의 갯수는 Table 1에 있는 부공간 축차과정에서의 이론적인 계산량을 이용하여 대략적으로 예측할 수 있다. 만약 a 개의 고유치를 x 개의 블록으로 나누어 계산한다고 하자. 그러면 한 블록당 총 축차벡터의 갯수 q 는 $q = 2a/x$ 의 관계가 성립한다. 또한 각 블록 당 평균 축차회수를 c_{avg} 라 하자. 그러면 블록 알고리즘의 총 계산량은 다음과 같다.

$$\begin{aligned} N.T. = x\left(\frac{1}{2}nm^2 + \frac{3}{2}nm\right) + \frac{2an}{x}(2m+1)x \\ + c_{avg}x\left(\frac{4mna}{x} + \frac{6na}{x} + \frac{8na^2}{x^2}\right) \end{aligned} \tag{6}$$

첫번째 항은 강성행렬의 분해, 두번째 항은 초기 리츠벡터의 생성, 세번째 항은 부공간 축차의 계산량을 각각 의미한다. 총 계산량이 최소가 되는 경우는 다음과 같다.

$$x = \sqrt{\frac{16c_{avg}a^2}{m^2+3m}} \tag{7}$$

여기서 특기할 점은 Table 1에서 m 은 평균 밴드폭의 반이지만 실제 사용된 해석기가 성긴(sparse) 해석기이므로 성긴 해석기의 특징상 해석과정에서 0으로 남는 행렬의 성분에 대해서 계산하지 않는 것을 고려할 때 m 은 실제 계산에서 사용되는 $\mathbf{K}$ 의 성분의 평균값으로 바꿔어야 한다.

$$m = \frac{\text{Number of total nonzero terms of } \mathbf{K}}{\text{Number of degree of freedoms}} \tag{8}$$

이때 강성행렬 $\mathbf{K}$의 대칭성을 고려하여 0이 아닌 성분의 갯수를 구해야 한다. 또한 실제 프로그램에서는 수치분해 후에 실제로 값을 갖는 부분을 예측하는 기호분해(symbolic factorization) 과정 다음에 m을 구해야 한다.

또한 위의 계산량에는 고정으로 인한 계산의 감소효과나 직교화로 인한 계산의 증가효과가 적용되지 않았다. 그러나 일반적으로 고정으로 인한 계산량의 감소효과가 훨씬 크기 때문에 고정의 효과만 고려해도 무방하다. 고정으로 인한 감소효과는 문제의 종류나 구하고자 하는 고유치의 갯수에 따라 달라지게 된다. 그러나 수치경험에 의하면 대략 30%~40% 정도 계산량의 감소효과가 있다. 평균적으로 35%의 감소효과를 적용하면 적절한 블록의 갯수는 다음과 같이 쓸 수 있다.

$$x=\sqrt{\frac{10.4c_{avg}a^2}{m^2+3m}} \tag{9}$$

또한 평균적인 축차회수도 고정 알고리즘과 마찬가지로 문제나 고유치의 갯수에 의존하지만 수정된 리츠벡터를 부공간 축차법의 초기벡터로 사용할 경우 경험적으로 보통 5~9회면 각 블록에서 수렴하므로 대략적으로 $c_{\text{avg}}=7$로 설정하였다.

3.3.5 수렴한 고유벡터와의 직교화

블록 알고리즘에서 중요한 과정 중의 하나는 이미 수렴한 고유벡터와의 직교화이다. 그 이유는 만약 이전 블록에서 수렴한 고유벡터와 직교화가 제대로 되지 않는다면 새로운 블록에서 만들어지는 모든 축차벡터가 이전 블록에서 수렴한 고유벡터로 다시 수렴하기 때문이다.

직교화를 확실하게 하기 위해 블록안에서 매 축차마다 수렴된 고유벡터와 직교화를 하며 또한 한 블록에서 다음 블록으로 넘어간 후 새로 만들어지는 초기벡터에 대해 다시 직교화를 한다.

Fig. 1은 개선된 블록 알고리즘의 전체적인 계산 흐름을 보여준다.

4. 수치예제

4.1 일반 사항

개선된 알고리즘의 성능을 검증하기 위하여 특징이 있는 몇 개의 문제에 적용하여 개선사항 별로 결과를 분석하였다. 적용한 문제에서 실제 계산에 참여하는 평균 항의 갯수 m은 Table 4에 있고 식 (9)를 이용하여 적절한 블록의 갯수를 계산하였다. 또한 Block+shift와 Block+shift+locking인 경우 초기벡터는 Bathe[1]가 제안한 초기벡터를 사용하였고 나머지 경우는 개선된 초기벡터를 사용하였다.

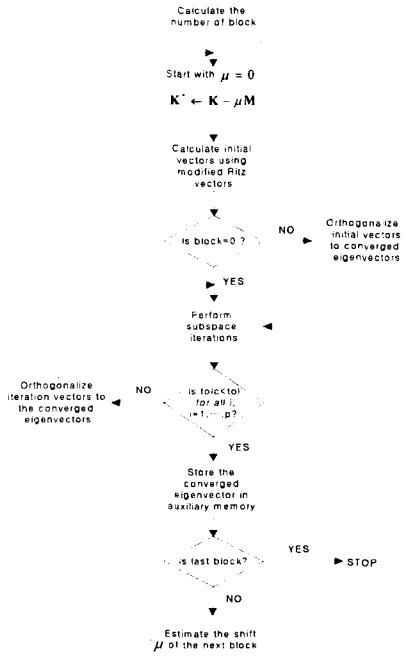


Fig. 1 Flow chart of the improved block subspace algorithm

Table 4 The aberage number of components of $\mathbf{K}$ used for actual operations

	Test problems		
	Plate	Frame structure	Automobile hood
Average number of components (m)	114	27	298

4.1.1 중근이 많이 있는 경우 성능 평가

모든 모서리가 단순지지된 평판에 적용하여 보았다. 사용된 재료는 탄성계수가 200GPa 이고 밀도가 7860kg/m^3 이며 두께가 0.01m 로 일정하기 때문에 많은 기하학적 대칭성을 가지고 있다.

Fig. 2는 기본적인 부공간 축차법과 비교하여 개선사항에 따른 시간절약의 정도를 나타내고 있다. 예외적인 경우도 있으나 대부분 블록과 제안

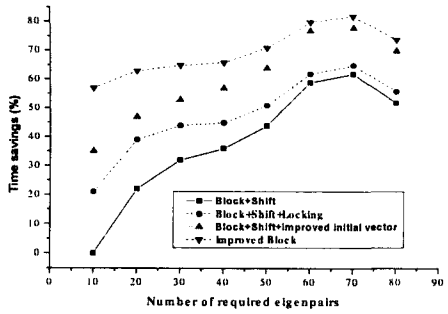
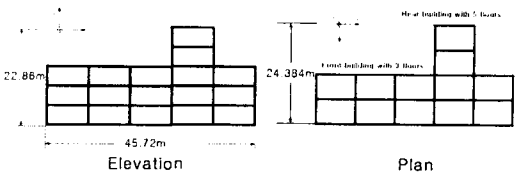


Fig. 2 Computational savings of modifications for the plate problem : comparing with the basic subspace iteration method

된 이동값을 사용한 알고리즘에서 가장 크게 시간절약이 되었으며 그 다음이 개선된 초기벡터를 적용하였을 경우이고 고정의 효과는 상대적으로 적었다. 물론 이런 모든 방법들을 결합한 개선된 방법의 시간절약이 가장 크다. 또한 중근이 많이 있는 문제임에도 불구하고 4가지 방법 모두 구하고자 하는 고유치와 고유벡터를 정확하게 구할 수 있었다. 이것은 Bathe가 제안한 초기벡터는 강성행렬의 대각항과 질량행렬의 대각항의 비를 이용해서 가능한 모든 고유벡터를 가진시키도록 만들어진 방법이기 때문이다. 그러나 이 방법은 초기벡터에 의해 만들어지는 부공간과 구하고자 하는 고유벡터에 의해 만들어지는 부공간 사이에 차이가 크기때문에 축차회수가 많아지고 이 때문에 계산시간이 오래걸리는 단점이 있다. 반면 개선된 초기벡터는 Bathe가 제안한 초기벡터의 장점을 가지고 있으면서 축차회수도 Bathe가 제안한 방법에 비해 적어지므로 시간을 절약하는 효과도 얻을 수 있다.

4.1.2 밴드폭이 작은 경우 성능 평가

Fig. 3은 3차원 프레임 구조이다. 빔요소로만으로 이루어져 있기때문에 자유도에 비해서 밴드폭이 작고 실제 계산되는 **K** 의 성분의 갯수도 매우 작다. 이런 경우 식 (9)로부터 계산된 블록의 갯수는 많아지게 되며 사용되는 주메모리의 양은 현저히 줄어들게 된다. Fig. 4를 보면 80개의 고유치를 구할 경우 블록은 24개가 되며 일반적으로 초기벡터를 구하고자 하는 고유치의 두 배로 잡는 란초스 벡터를 부공간 축차의 초기벡터로 이용하는 경우에 비해 대략 80%정도 주메모리의 사용량을 절약할 수 있다.



Column in Front Building: A =0.2787m^2,I =8.631Ç10^{-3}m^4

Column in Rear Building : A =0.3716m^2,I =1.079Ç10^{-4}m^4

All Beams into X-Direction: A =0.1858m^2,I=6.473Ç10^{-3}m^4

All Beams into Y-Direction: A =0.2787m^2,I=8631Ç10^{-3}m^4

E=20GPa , ρ =7800 kg/m^3

Fig. 3 Three-dimensional frame structure

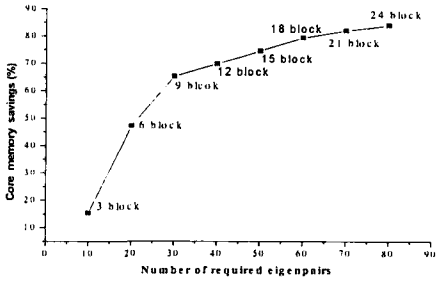


Fig. 4 memory savings of the block algorithm

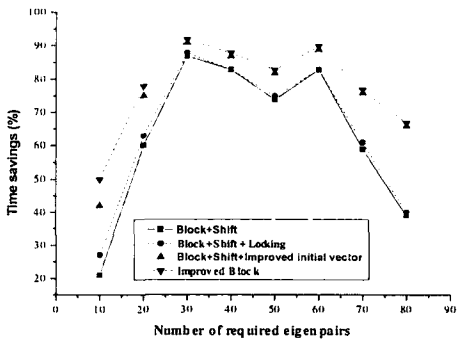


Fig. 5 Computational savings of modifications for the frame structure problem : comparing with the basic subspace iteration method

Fig. 5는 기본적인 부공간 축차법과 비교하여 개선사항에 따른 시간절약의 정도를 나타내고 있다. 빌딩구조인 경우 10개의 고유치를 구할 때부터 이동이 적용되기 때문에 블록과 제안된 이동값을 사용하는 경우가 가장 시간절약이 크게 되었으며 그 다음이 개선된 초기벡터를 사용한 경우였다.

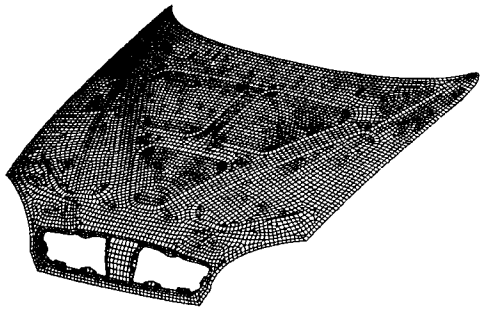


Fig. 6 Finite element model of an automobile hood

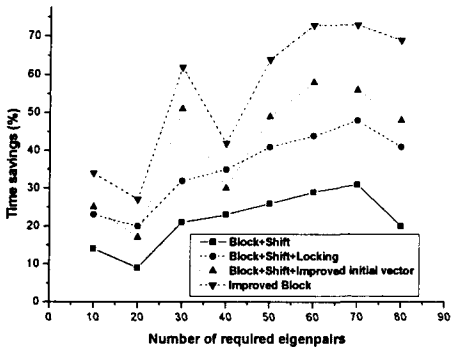


Fig. 7 Computational savings of modifications for the automobile hood problem : comparing with the basic subspace iteration method

4.1.3 대형 문제에 적용할 경우 성능 평가

Fig. 6은 60,000 자유도 정도로 모델링 된 자동차 후드로 다양한 요소와 재료가 사용된 실제적인 문제이다. Fig. 7은 기본적인 부공간 축차법과 비교하여 개선사항에 따른 시간절약의 정도를 나타내고 있다. 이 문제는 40번째 고유치를 구할 때부터 이동을 하게 되지만 이동을 해도 성능이 이전 문제처럼 크게 개선되지 않는다. 그 이유는 강성행렬의 분해로 인한 시간이 이전 문제들에 비해 상대적으로 많이 걸리기 때문이다. 이 문제의 경우 개선된 초기벡터를 사용한 경우가 가장 크게 시간을 절약하고 고정 알고리즘도 시간절약에 크게 영향을 미치고 있다. 물론 이 모든 방법들을 결합한 개선된 방법은 구하고자 하는 고유치의 갯수에 상관없이 가장 좋은 성능을 유지하는 것을 확인할 수 있다.

5. 결 론

기존의 부공간 축차법이 가지고 있던 비효율성을 개선하기 위하여 고정, 이동, 적절한 초기벡터, 블록 알고리즘을 결합한 개선된 알고리즘을 제안하였다. 또한 다양한 알고리즘들이 결합되는 과정에서 발생할 수 있는 문제점과 이에 대한 적절한 해결방안을 제시하였다.

제안된 알고리즘은 수정된 리츠벡터와 적절한 이동값을 갖는 준정적 리츠벡터를 초기벡터로 사용하며 블록 알고리즘을 바탕으로 하고 있기 때문에 중근이 있거나 고유치가 모여있는 문제도 고유치를 빠뜨리지 않고 구할 수 있으며 블록의 갯수에 따라 주메모리의 사용량을 줄일 수 있다. 동시에 기존의 기본적인 부공간 축차법의 계산시간을 50%에서 최대 90%까지 줄일 수 있으며 어떤 문제에 대해서도 성능이 유지되는 강건한 방법이다.

참고문헌

(1) Bathe, K. J., 1982, *Finite Element Procedures in Engineering Analysis*, Prentice-Hall, Englewood Cliffs, New Jersey.

(2) Bertolini , A. F. and Lam, Y. C., 1998, "Accelerated Subspace Iteration Using Adaptive Multiple Inverse Iteration," *Computers & Structures*, Vol. 66, pp. 45~57.

(3) Qian, Y. and Dhatt, G., 1995, "An Accelerated Subspace Method for Generalized Eigenproblems," *Computers & Structures*, Vol. 54, pp. 1127~1134.

(4) Wilson, E. L. and Itoh, T., 1983, "An Eigensolution Strategy for Large Systems," *Computers & Structures*, Vol. 16, pp259~265.

(5) Jennings, A. and Stewart, W. J., 1981, "A Simultaneous Iteration Algorithm for Real Matrices," *ACM Transactions on Mathematical Software*, Vol. 7, pp. 184~198.

(6) Akl, F. A., 1982, "Acceleration of Subspace Iteration," *International Journal of Numerical Methods in Engineering*, Vol. 18, pp. 583~589.

(7) Bathe, K. J. and Ramaswamy, S., 1980, "An Acceleration Subspace Iteration Method," *Computer Methods in Applied Mechanics and Engineering*, Vol. 23, pp. 313~331.

(8) Gu, J., Ma, Z-D. and G. M. Hulbert, 2000, "A New Load-Dependent Ritz Vector Method for Structural Dynamics Analyses : Quasi-Static Ritz Vectors," *Finite Elements in Analysis and Design*, Vol. 36, pp. 261~278.

(9) Jiaxian, X., 1989, "An Improved Method for Partial Eigensolution of Large Structures," *Computers & Structures*, Vol. 32, pp. 1055~1060.

(10) Trefethen, L. N. and Bau, D., 1997, *Numerical Linear Algebra*, SIAM, Philadelphia.