1852 대한기계학회논문집(A) 제21권 제11호, pp. 1852~1861, 1996

〈논 문〉

부공간축차법의 효율향상을 위한 연구

이 병 채* · 오 승 환**

(1997년 1월 17일 접수)

A Study on the Development of an Efficient Subspace Iteration Method

Byung-Chai Lee and Seung-Hwan Oh

Key Words: Eigenvalue Problem(고유치 해석), Subspace Iteration Method(부공간축차법), Lanczos Method(란초스방법)

Abstract

An enhanced subspace iteration algorithm has been developed to solve eigenvalue problems reliably and efficiently. Basic subspace iteration algorithm has been improved by eliminating recalculation of converged eigenvectors, using Krylov sequence as initial vectors and incorporating with shifting techniques. The number of iterations and computational time have been considerably reduced when compared with the original one, and reliabililty for catching copies of the multiple roots has been retained successfully. Further research would be required for mathematical justification of the present method.

기호설명

n : 구할 고유치의 갯수
m : 부공간의 크기
x : 서로 독립인 r개의 초기벡터
$\bar{y}_l$: Krylov 점렬과 x을 이용한 m개의 초기벡터
K : 강성행렬
M : 질량행렬
μ_s : Shift point
ω : 시스템의 고유치
ϕ 시스템의 고유벡터

1. 서 론

현재 유한요소해석에서 고유치 해석에 소요되는 시간은 선형 정적해석에 비해 막대하며, 이는 유한요소해석을 통한 대화식 설계작업에 있어 상당한 저해요인으로 간주되고 있다. 따라서 고유치 해석의 계산효율 개선을 위한 연구가 요청되고 있으며, 세계적으로도 상당한 연구가 이루어지고 있다. 본 논문에서는 당해 연구팀에서 개발, 이용하고 있는 기존 유한요소해석 프로그램에 현재까지 알려진 우수한 방법들을 추가하여 수치적으로 그 특성을 분석하고 아울러 개선방안을 도출한다.

고유치 해석은 근본적으로 축차에 의해 이루어지고 있으며, 전 시스템의 고유치보다는 그중 극히 작은 수의 일부 고유치들에 관심이 있는 경우가 대부분이며, 이에 적합한 부공간축차법(subspace iteration method)[1]이나 Lanczos 축차 해법[2]이 가장 많이 사용되고 있다.

부공간축차법은 공학해석에서 가장 많이 이용되고 있는 고유치 해법이며, 대형 시스템의 일부 고유치와 해당 고유벡터를 구하는 방법이다. 부공간축차법은 근본적으로 다음 세 과정을 반복하여 고유치를 계산한다.[1] 구하고자 하는 고유치 수보다 적당히 큰 부공간 크기를 정하고 초기벡터를 근사한다.[2] 동시역축차기법을 이용하여 현재 근사공간

*회원, 한국과학기술원 기계공학과

에서 가장 좋은 근사고유치와 고유벡터를 구한다.[3] 전 과정을 반복수행하며 수렴여부를 검토하여 원하는 고유치가 모두 계산되었는지 확인한다. 이 방법의 근원은 동시벡터축차법,[3,4] Ritz 기초벡터 산출법[5] 등으로 볼 수 있고 안정적으로 고유치를 잘 구할 수 있다는 점이 큰 장점이나 구하려는 고유치 수가 많아질 때 계산효율이 급격히 감소한다는 점이 단점이다. 최근 수렴을 가속화 시키기 위한 연구,[6,7] 초기벡터를 효율적으로 구하는 연구,[8] 란초스방법과의 복합을 위한 연구[9,10] 등도 이루어지고 있다.

란초스방법은 변환에 의해 행렬을 세항 대각행렬(tri-diagonal matrix)로 만들고, 이로부터 고유치와 고유벡터를 계산하는 방법이다. 실제 이용시에는 컴퓨터 실수표현의 한계로 인해 란초스벡터의 수직성이 훼손된다는 것이 난점으로 이를 해결하기 위한 여러가지 방법들이 제안되었다.[11~13] 또 중근 고유치를 놓칠 가능성이 있다는 점, 중근아닌 고유치가 복사될 수 있다는 점, 수렴판정이 어렵다는 점, 축차벡터수가 부공간축차법에 비해 상당히 많다는 점 등이 이 방법의 단점으로 언급되고 있다. 대각 질량행렬의 문제에 대해 란초스방법과 부공간축차법의 효율을 비교한 연구[14]에서는 란초스방법의 효율이 상대적으로 우수하고 구하는 고유치 개수가 많아질수록 그 차이가 커지는 것으로 나타났다. 따라서 신뢰성과 효율성을 동시에 만족시키기 위해 두 방법을 복합하는 방법을 고려할 수 있겠다.

본 논문에서는 일차적으로 부공간축차법의 성능개선을 시도하였으며, 다음과 같은 몇가지 사항을 고려하여 괄목할 만한 효율증대를 이룩하였으며, 계속적인 개선과 안정성 향상을 도모하고 있다.

2. 고유벡터 수렴정보의 이용

본 논문의 기초인 부공간축차법의 기본 알고리듬은 Bathe의 논문[1]에 잘 기술되어 있다. 이를 요약하면 다음과 같다.

(1) 부공간의 적정크기 산정 : $m=min(n*2,\ n+8)$

여기서, n은 구할 고유치 수를, m은 부공간 크기를 나타낸다.

(2) 초기벡터 계산 : m개의 초기벡터를 모은 행렬을 X라 할 때, 실제 축차계산 시작시 요구되는 벡터는 $y=Mx$이므로 y벡터를 모은 행렬 Y를 예측한다.

여기서, M은 구조질량행렬이다. 일반적인 초기 Y행렬 예측방법은 다음과 같다.

(1) Y행렬의 첫열은 M행렬의 대각선 항으로 한다.

(2) Y행렬의 다음 열들은 차례로 단위벡터로 정의한다. 각 단위벡터의 1인 항은 k_{ii}/m_{ii}비가 작은 순서에 따라 차례로 정한다.

(3) 벡터를 변환한다.

$$K\bar{X}_{k+1}=Y_k$$

(4) 부공간 고유치문제를 만든다.

$$K_{k+1}=\bar{X}^T_{k+1}Y_k$$
$$\bar{Y}_{k+1}=M\bar{X}_{k+1}$$
$$M_{k+1}=\bar{X}^T_{k+1}\ \bar{Y}_{k+1}$$

(5) 부공간 고유치문제를 푼다.

$$K_{k+1}\ Q_{k+1}=M_{k+1}Q_{k+1}\Lambda_{k+1}$$

(6) 벡터를 수정한다.

$$Y_{k+1}=\bar{Y}^T_{k+1}\ Q_{k+1}$$

(7) 수렴을 체크하여 수렴하였으면 다음으로, 그렇지 않으면 과정 (3)으로 되돌아간다.

(8) 필요에 따라서는 Sturm 점렬체크를 통해 원하는 고유치가 모두 얻어졌는지 검토한다.

부공간축차법에서 시간이 많이 소요되는 작업은 과정 (3)의 연립방정식 풀이와 과정 (4)에서의 질량행렬-벡터 곱계산과정으로 대체로 80% 이상의 시간이 여기에 소요된다. 그런데 고유벡터가 수렴된 이후에도 그 벡터에 대한 연립방정식 풀이와 질량행렬-벡터 곱계산을 반복하도록 알고리듬이 구성되어 있다. 따라서 본 연구에서는 수렴된 고유벡터뿐 아니라 중간계산과정에 필요한 벡터들을 저장해 둠으로써 무의미한 반복계산을 피하도록 프로그램하였다. 결국 기억용량을 크게 하고 계산 효율을 올린 셈인데 실제 구하는 부공간 수가 시스템 자유도에 비해 매우 작은 점과 최근 메모리 가격의 저렴화 경향을 고려하면, 이 편이 보다 나은 선택이라 할 수 있다.

3. Krylov 점렬을 이용한 초기벡터 산출

부공간축차시 많은 계산이 각 축차에서 이루어지므로 축차회수를 줄이는 것이 전체 계산시간 단축

에중요하다. 수렴을 가속화하는 여러가지 방법[6,7,10]이 있으나 다양한 문제에 안정적으로 적용할만한 방법은 드물다. 참고문헌[1]에 언급된 바와 같이 해공간을 포함하는 공간의 기저벡터들로 초기 벡터를 잡으면 한번만에 해를 구할 수 있다. 부공간축차법의 효율은 초기 부공간이 해공간에 얼마다 근접해 있느냐에 따라 결정되므로 초기벡터를 잘 잡는 것이 효율향상을 위해 매우 중요하다. 더구나, 구하는 고유벡터가 초기 부공간에 수직이 되면 그 고유벡터는 축차를 통해 구할 수 없게 된다.

초기벡터를 개선하여 효율을 개선하기 위한 시도로 임의벡터를 이용한 방법, 란초스벡터를 이용한 방법, 동적축약(dynamic condensation, Guyan reduction)에 근거한 방법 등이 있으며, 이중 동적축약에 의한 방법이 효율적인 것으로 알려져 있다[8]. 그러나 동적축약에 의한 방법은 부공간축차법과는 계열이 다른 방법이고 또 축약을 시행하면 시스템행렬의 Book-keeping 이 복잡해져 전용 프로그램을 개발해야 하는 난점이 있다. 또 이 경우 시스템행렬처리에 시간이 많이 소요되는 경우가 발생하므로 대형 유한요소 프로그램에서의 사용에는 아직 문제가 있다. 본 연구에서는 Lanczos 방법에서 이용하고 있는 Krylov 점렬을 개선하여 초기벡터를 산출하고, 부공간축차를 행하는 방법을 택하였다.[9]

Krylov 점렬은 출발벡터를 x, $S=K^{-1}M$라 할 때

$$\{x,\ Sx,\ S^2x, \cdots,\ S^nx,\ \cdots\}$$

로 정의된다. 이 점렬을 이용할 때 중근을 구하지 못 하는 경우가 종종 있다. 그러나 다음과 같이 출발벡터 x를 서로 독립인 r개의 벡터로 택하여 Krylov 점렬의 적당항까지를 초기벡터로 택하면 r개까지의 중근을 모두 구할 수 있게 된다.[9]

$$\bar{y}_i = Sy_{i-1}$$

여기서,

$$y_{i-1}=\begin{cases} \bar{y}_{i-1}+x_i & i=1,\ 2,\ \cdots,\ r \\ \bar{y}_{i-1} & i=r+1,\ r+2,\ \cdots,\ m \end{cases}$$

이렇게 초기벡터를 구하면 다음과 같은 이유에 의해 중근을 항상 구할 수 있게 된다.

시스템행렬의 크기를 p라 하고, 시스템의 고유치와 고유벡터를 각각 ω_i, ϕ_i라 한다. 기술의 편의상 고유치중 가장 작은 값이 t 중근이라고 가정한다. 이 중근에 해당하는 고유벡터를 ϕ_1, ϕ_2, $\cdots$, ϕ_t라 정의하면 이 고유벡터들은 t차원 부공간 E_t를 구성하게 된다. 이때 모든 출발벡터 x는 고유벡터가 p차원 공간의 기저임을 고려하면 다음과 같이 표현될 수 있다.

$$x_i=\alpha^{(i)}\tilde{\phi}^{(i)}+\sum_{j=t+1}^{p}\alpha_j^{(i)}\phi_j,\quad i=1,\ 2,\ \cdots,\ t$$

여기서,

$$\alpha^{(i)}\tilde{\phi}^{(i)}=\sum_{l=1}^{t}\alpha_l^{(i)}\phi_l,\quad i=1,\ 2,\ \cdots,\ t$$

의 간략 표현이다. 이는 $\tilde{\phi}^{(i)}$가 E_t 공간의 한 벡터임을 나타낸다. 이 때, 임의의 출발벡터 x_t의 Krylov 점렬은 다음 식에 의해 고유치, 고유벡터로 표현될 수 있다.

$$K^{-1}Mx_i=\alpha^{(i)}\omega_1^2\tilde{\phi}^{(i)}+\sum_{j=t+1}^{p}\alpha_j^{(i)}\omega_j^2\phi_j$$

$$(K^{-1}M)^s x_i=\alpha^{(i)}\omega_1^{2s}\tilde{\phi}^{(i)}+\sum_{j=t+1}^{p}\alpha_j^{(i)}\omega_j^{2s}\phi_j$$

$$s=m+1-i,\quad i=1,\ 2,\ \cdots,\ t$$

모든 초기벡터 $\bar{y}_i$는 전술한 Krylov 점렬벡터의 선형결합으로 표현된다. 출발벡터 한개, 즉 x_0로부터 초기벡터를 만들면 모든 $\bar{y}_i$는 $\tilde{\phi}^{(i)}$, ϕ_{r+1}, ϕ_{r+2}, $\cdots$, ϕ_p의 선형결합이 된다. 그러므로 이 경우 초기벡터 수를 아무리 늘린다 해도 E_t 공간의 나머지 r-1 차원은 고려되지 않음을 알 수 있다. 실제 문제에서 표준 란초스방법이 중근을 종종 구하는 것은 컴퓨터의 끝자리 오차(round-off error) 때문인 것으로 밝혀져 있다. 그러나 이 경우 초기벡터 수를 상당히 많게 해야 중근을 구할 수 있다. 반면에 여러 개의 출발벡터를 사용하면 사용한 개수만큼의 벡터로 E_t 공간을 표현하게 되어 중근을 놓칠 가능성이 현저히 줄게 된다. 또, 중근을 갖지 않은 시스템의 경우에도 여러 개의 출발벡터를 고려함으로써 보다 다양한 자유도를 활성화 시킬 수 있고, 이에 따라 고유치를 놓칠 가능성이 훨씬 줄게 된다.

Krylov 점렬 이용시 수치적 안정성과 효율을 높이기 위해 주어진 벡터들이 M행렬에 대해 수직이 되도록 Gram-Schmidt 수직화과정을 거쳤다. 이 방법 이용시 부공간의 크기는 구할 고유치 수의 두 배 이상으로 하였다.[9] 가장 이상적인 값은 앞으로 다양한 수치실험을 통해 정해야 할 것으로 생각된다. Lanczos 방법에서 여러 개의 출발벡터를 이용하여 Krylov 점렬을 구하는 방법은 약간의 차이는 있으나 블록(block) Lanczos,[15] 폭(band) Lanczos,[11]

부공간(subspace) Lanczos 방법[16] 등으로 이미 알려져 있는 방법이다.

4. 미수렴 고유치의 효율적 수렴기법

Krylov 점렬을 부공간축차법의 초기벡터로 이용할 경우 부공간축차 초기의 고유치들도 상당히 정확함을 수치경험을 통해 알 수 있었다. Krylov 점렬을 이용할 때 부공간축차를 수회만 수행하여 얻어진 고유치와 고유벡터를 이용하여 빨리 수렴시키는 방법을 고안하였다. 해 근처의 고유치를 수렴시키는 방법으로 여러 가지 기법들이 고려될 수 있으나 본 연구에서는 역축차기법(inverse iteration method)과 쉬프트(shift)한 후 다시 부공간 축차를 시행하는 방법을 고려하였다. 역축차기법에서도 수렴시키려는 고유치가 다른 값에 접근하는 것을 막고 수렴속도를 증대시키기 위해 현재 근사고유치로 고유치문제를 쉬프트시켰다. 역축차기법의 알고리듬은 다음과 같다.

Step 0. 수렴시켜야 할 고유치번호 추출

Step 1. 쉬프트

(1) 강성행렬 쉬프트 : $\tilde{K}=K-\mu_t M$
여기서, μ_t는 수렴시켜야 할 고유치이다.

(2) 쉬프트된 강성행렬 $\tilde{K}$를 Crout 분해한다.

Step 2. 역축차기법

(1) $y_k=Mx_k$

(2) $\tilde{K}\bar{x}_{k+1}=y_k$

(3) $\bar{y}_{k+1}=M\bar{x}_{k+1}$

(4) $\bar{\mu}_t=\bar{x}_{k+1}{}^T y_k/\bar{x}_{k+1}{}^T\bar{y}_{k+1}$

(5) $y_{k+1}=\bar{y}_{k+1}/(\bar{x}_{k+1}{}^T\bar{y}_{k+1})^{1/2}$

(6) $\bar{u}_t$가 수렴하였으면 Step 3으로 그렇지 않으면 k를 $k+1$로 대치하고 (1)로 되돌아 간다.

Step 3. 모든 고유치를 수렴시켰으면 과정을 끝내고 그렇지 않으면 새로 수렴시킬 고유치를 μ_t로 하여 Step 1로 되돌아 간다.

이 방법을 적용할 때 수렴되지 않은 고유치가 많으면 강성행렬 분해를 여러 차례 수행해야 한다. 따라서 분해에 소요되는 시간이 부공간축차 1회 소요시간에 비해 긴 경우에는 효율이 떨어진다. 이를 해결하기 위해 본 논문에서는 Krylov 점렬로부터 구한 초기벡터로부터 2회의 부공간축차를 행하고 이 때 n번째 고유치부터 시작하여 다음 조건을 만족하는 가장 큰 쉬프트 값을 선택하였다. 부공간축차로부터 얻은 근사고유치를 λ_t라 하면 쉬프트 μ_s는

$$\mu_s=\frac{\lambda_n+\lambda_{n-1}}{2}$$

로 하며, 이때 $1.01\lambda_{n-1}\leq\mu_s\leq 0.99\lambda_n$이면 n을 1씩 감소시켜 가며, 이 조건을 만족하는 μ_s를 찾는다. 쉬프트한 다음 부공간축차법을 연이어 수행하면 수렴속도가 증대된 방법을 만들 수 있다. 이 방법은 구조행렬 분해시간이 부공간축차시간에 비해 비교적 짧은 경우 매우 효과적이다. 따라서 10개 이하의 고유치를 구하는 문제에는 큰 개선효과가 나타나지 않았다.

5. 계산결과 및 토의

전술한 방법들의 성능을 비교하기 위해 주로 건물모델과 평판구조의 유한요소 자료를 중심으로 몇 가지 문제를 풀어 보았다. 문제의 개요를 Table 1에 수록하였다. 문제 1, 2, 3은 보요소로만 이루어진 건물모델을 푼 것이며, 문제 4는 4개의 변이 단순지지된 평판문제를 푼 것이다. 각 문제에 대한

Table 1 Statistics of example problems

	Problem 1	Problem 2	Problem 3	Problem 4
No. of element	972	818	2940	2500
No. of node	534	608	1516	2601
No. of d.o.f	3024	3600	9000	7399
Profile	494748	235000	1146060	1121115
Assemble time(sec)	1.2	0.71	13.7	8.29
Decomposition(sce)	29.9	6.92	78.9	55.3

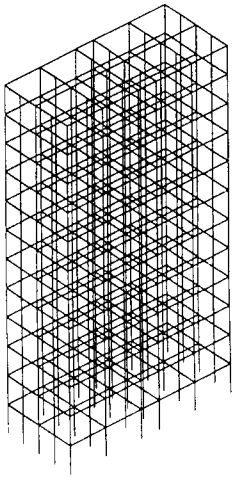

Fig. 1 Building structure consist of beams (1)

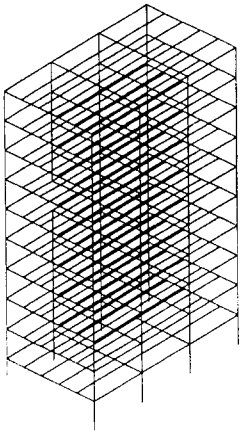

Fig. 2 Building structure consist of beams (2)

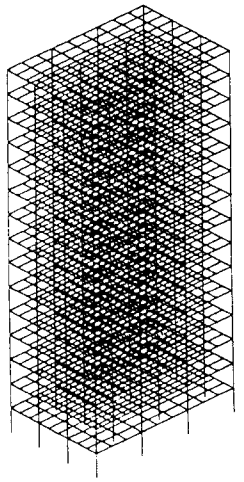

Fig. 3 Building structure consist of beams (3)

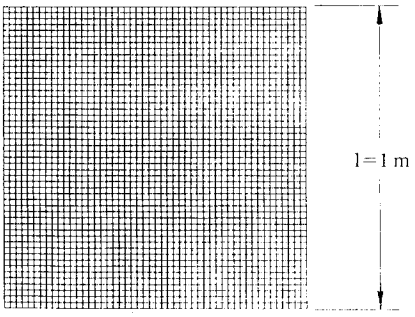

Fig. 4 Simply supported plate

유한요소 모델의 형상을 Figs. 1~4에 그렸다.

본 논문에서의 계산은 펜티엄 PC 에서 수행하였으며 방법간의 비교분석을 위해 중요 파라메터를 바꾸어 가며 작업하였다. 작업결과를 Tables 2~5에 수록하였는데, 방법은 Pa(p)-Mb(n) 형식으로 정의하였다. 여기서, a는 문제번호를 p는 구할 고유치의 수를 나타낸다. 또, b는 적용한 방법의 번호로 1(n)은 부공간의 크기가 n인 보통의 부공간축차법을, 2(n)는 부공간의 크기가 n이고, 수렴된 고유벡터 재계산이 배제된 부공간축차법을, 3(n)은 부공간의 크기가 n인 초기벡터를 개선한 방법을 나타낸다. 또, 방법 4(n)은 부공간 크기를 n으로 한 Krylov 점렬 초기치로 2회 부공간 축차한 후 쉬프트과정을 거치고, 부공간축차법을 다시 적용한 방법이다. 고유치의 수렴은 각 고유치 상대오차가 0.00001 이하일 때 이루어진 것으로 판단하였다. 문제 1과 문제 2는 문제의 크기가 크지 않아 강성행렬, 질량행렬 등 작업파일을 PC의 램드라이브에 기록할 수 있었으나 문제 3의 경우에는 이것이 불가능하여 하드디스크에 작업파일의 일부를 기록하였다. 같은 문제를 풀 경우 하드디스크에 기록하는 방법의 계산시간이 30~50 % 정도 더 소요된다. 수렴오차에 따른 영향을 살펴보기 위해 수렴오차를 0.000000001 로 작게 하여 방법 2와 방법 4로 고유치 계산을 하였고, 그 결과를 Table 6에 수록하였다. 이 때 부공간의 크기는 비교를 위해 같게 하였다. 누락 고유치 발생여부를 판정하기 위해 Sturm 점렬체크법을 이용하였으며, 부공간축차법의 수렴오차 10^{-9}인 해를 기준으로 하여 정밀한 해를 구할 수 있는지 검토하였다. 계산한 여러 가지

Table 2 Time for eigen analysis, sec (Problem 1)

Problem	Method	Matrix multiply	Eigen analysis	Total CPU time
P1(10)	M1(18)	53.88	204.49	242.44
	M2(18)	43.45	162.91	200.87
	M2(20)	40.97	143.03	176.25
	M3(20)	33.98	123.92	162.58
	M4(20)	26.07	127.10	153.25
P1(20)	M1(28)	108.75	410.73	449.45
	M2(28)	80.15	296.76	335.43
	M2(40)	78.21	272.38	306.76
	M3(40)	81.84	302.25	340.97
	M3(54)	75.35	279.68	318.46
	M4(40)	58.57	233.82	266.11
P1(30)	M1(38)	235.03	889.35	928.30
	M2(38)	148.10	532.83	571.88
	M2(60)	150.49	525.37	559.25
	M3(60)	92.34	343.72	382.72
	M4(60)	82.68	322.13	354.49

Table 3 CPU time for eigen analysis, sec (Problem 2)

Problem	Method	Matrix multiply	Eigen analysis	Total CPU time
P1(10)	M1(18)	35.65	126.54	136.32
	M2(18)	29.92	104.24	114.08
	M2(20)	26.78	89.53	98.54
	M3(20)	15.31	56.41	66.24
	M4(20)	10.83	40.31	48.34
P(20)	M1(28)	131.08	460.44	470.27
	M2(28)	90.88	305.33	315.16
	M2(40)	88.79	287.53	296.54
	M3(40)	111.05	396.61	406.45
	M3(54)	54.55	203.66	213.55
	M4(40)	45.67	167.45	177.52
P2(30)	M1(38)	273.21	955.98	965.81
	M2(38)	171.35	545.19	555.02
	M2(60)	129.12	432.10	441.16
	M3(60)	177.49	636.14	646.09
	M3(85)	85.18	337.02	346.09
	M4(60)	107.37	393.11	401.18

Table 4 CPU time for eigen analysis, sec (Problem 3)

Problem	Method	Matrix multiply	Eigen analysis	Total CPU time
P3(10)	M1(18)	97.03	391.07	470.99
	M2(18)	83.27	338.45	418.31
	M2(20)	71.46	279.46	350.70
	M3(20)	71.13	330.05	409.97
	M4(20)	62.86	295.44	355.75
P(20)	M1(28)	368.29	1424.83	1504.69
	M2(28)	232.16	887.43	967.46
	M2(40)	252.10	909.18	980.48
	M3(40)	136.33	622.69	702.67
	M4(40)	102.36	370.03	430.45
P3(30)	M1(38)	822.72	3132.02	3212.04
	M2(38)	609.16	2118.48	2198.50
	M2(60)	669.54	2352.13	2423.59
	M3(60)	347.22	1471.68	1551.76
	M4(60)	211.08	852.94	913.47

Table 5 CPU time for eigen analysis, sec (Problem 4)

Problem	Method	Matrix multiply	Eigen analysis	Total CPU time
P4(10)	M1(18)	79.52	282.54	354.66
	M2(18)	70.97	250.18	322.42
	M2(20)	79.75	279.90	352.08
	M3(20)	79.92	278.58	350.08
	M4(20)	64.22	302.97	375.09
P4(20)	M1(28)	182.38	639.23	711.34
	M2(28)	140.48	482.13	554.37
	M2(40)	167.98	609.23	681.52
	M3(40)	122.35	447.97	520.25
	M4(40)	123.55	529.15	601.32
P4(30)	M1(38)	318.11	1172.72	1244.89
	M2(38)	226.91	791.10	863.32
	M2(60)	273.30	1056.00	1128.50
	M3(60)	176.80	679.31	751.60
	M4(60)	150.99	585.56	657.85

Table 6 CPU time for eigen analysis with convergence tolerance 10^{-9}, sec (Problem 4)

Problem	Method	Matrix Multiply	Eigen analysis	Total CPU time
P1(10)	M2(20)	54.25	190.26	222.50
	M4(20)	31.37	141.65	173.90
P1(20)	M2(40)	119.04	415.40	447.70
	M4(40)	68.60	237.26	309.56
P1(30)	M2(60)	192.10	673.72	706.07
	M4(60)	92.50	372.12	404.42
P2(10)	M2(20)	38.07	127.26	135.28
	M4(20)	14.90	60.52	68.49
P2(20)	M2(40)	128.85	420.90	428.97
	M4(40)	53.47	202.78	210.86
P2(30)	M2(60)	173.13	579.30	587.43
	M4(60)	122.06	456.65	464.67
P3(10)	M2(20)	99.97	348.77	409.20
	M4(20)	63.55	301.98	362.34
P3(20)	M2(40)	340.14	1164.64	1225.23
	M4(40)	132.93	561.28	621.70
P3(30)	M2(60)	855.37	2927.75	2988.49
	M4(60)	276.39	1107.79	1168.32
P4(10)	M2(20)	100.18	352.29	423.04
	M4(20)	70.84	326.09	423.04
P4(20)	M2(40)	212.14	768.29	839.21
	M4(40)	133.77	566.66	637.52
P4(30)	M2(60)	342.78	1321.34	1392.36
	M4(60)	151.32	586.05	656.91

경우에서 누락된 고유치는 없었으며, 본 논문에서 제안한 방법에 의한 고유치 값도 기준되는 해와 동일하였다.

문제 4는 얇은 판 문제로 정해와 비교하여 부공간축차법에 의한 30개의 고유치까지 누락된 고유치가 없었다.

계산결과를 분석해 보면 다음과 같은 결론을 얻을 수 있다.

첫째, 수렴된 벡터에 대한 재계산을 배제함으로써 일관된 효율 향상이 이루어졌다. 계산시간은 평균하여 26.1% 단축되었다. 이 때 축차회수도 증가하지 않았다. 이 감소효과는 계산하는 고유치가 많을수록 증가하는 경향을 보였다.

둘째, Krylov 점렬과 부공간축차법을 이용한 방법은 대부분 좋은 결과를 주었으나 고유치 순서가 축차에 따라 바뀌는 경우 오히려 나빠지기도 하였다. 부공간의 크기에 따라 방법의 효율이 달라지므로 부공간 최적크기를 예측하는 기법의 개발이 요구된다. 대체로 구하는 고유치 수의 2.7~2.8 배일 때의 결과가 가장 좋았다.

셋째, 상대 비교를 위해 부공간축차법의 부공간 크기를 복합방법의 부공간 크기와 같게하여 계산해 보았다. 대체적으로 부공간 크기를 키우면 약간 나은 결과를 주나 복합방법만큼의 효율 증대는 나타나지 않았다. 때로는 부공간 크기가 큰 경우가 오히려 계산 시간이 많이 소요되기도 하였다. 이로 보아 기본 부공간축차법에서 부공간의 크기는 참고문헌[1]에 기술된 식이 충분한 것으로 판단된다.

넷째, 수렴판정오차와 계산효율의 관계를 살펴보기 위해 결과가 우수한 것으로 나타난 M2 방법과 M4 방법을 같은 부공간 크기로 계산하였다. 표 6에 보이는 계산결과에서 판정기준을 올리면 모든 경우 새로운 방법의 효율이 우수하게 됨을 알 수 있다. 또, 두 방법의 계산시간 비를 평균해 보면 판정오차가 0.00001일 때 0.7275이었고 오차가 0.000000001일 때는 0.649이었다. 이로부터 수렴판정오차를 작게 하였을 때 기본 부공간축차법이 본 연구의 복합방법보다 큰 영향을 받음을 알 수 있다. 즉, 수렴판정오차를 작게 했을 때 기본 부공간축차법은 계산시간이 크게 증가한 반면, 새로운 방법은 상대적으로 작게 늘어남을 알 수 있다.

다섯째, Krylov 점렬과 부공간축차법, 쉬프트후 다시 부공간축차법을 이용한 방법이 가장 우수하였으며, 평균하여 45.7%의 계산시간이 감소하였다. 부공간 축차법만의 결과가 아주 우수한 경우, 행렬분해시간이 많이 소요되는 경우, 구하는 고유치 수가 비교적 작은 경우-P4(10), P4[20]에는 오히려 나쁜 결과를 주기도 했다. 그러나 이 경우에도 수렴판정기준을 높이면 복합방법이 가장 좋은 결과를 준다. 부공간축차법의 계산시간이 수렴판정오차나 문제특성에 따라 민감하게 변하고, 고유 공간크기가 충분치 않을 경우 누락 고유치 발생 가능성이 있다는 점 등을 고려하면 새로운 기법이 신뢰성과 효율, 정확성 면에서 가장 우수한 방법이라고 결론지을 수 있다.

6. 결 론

본 논문에서는 동적 구조해석이나 선형 좌굴해석에 필수적인 고유치 해석방법의 효율을 개선하기 위해 부공간축차법과 Krylov 점렬 이용방법을 효과적으로 결합시켜 새로운 복합방법을 개발하였다.

부공간축차법 자체의 효율개선을 위해 수렴된 고유벡터에 대한 재계산을 배제하였다. 이를 통해 20~30% 정도의 효율향상을 이루었다. Krylov 점렬을 이용한 초기벡터 개선과 쉬프트를 통해 역시 20~30%의 효율 향상을 이루었다. 아울러 다양한 수치 예로부터 새로운 방법이 수렴판정기준에 보다 덜 민감하고 중근을 잘 구할 수 있으며, 경험적 요소가 별로 필요없는 강건한 방법임을 알 수 있었다.

전술한 기법들의 종합적인 효과로 현재 개발된 프로그램은 효율성과 신뢰성 면에서 매우 우수한 계산 결과를 주고 있으며, 이에 대한 이론적 보완과 실제 문제에서 발생할 수 있는 다양한 상황에의 대처가 잘 이루어지면 지금까지 발표된 고유치 해법중 가장 좋은 실제적 방법의 하나가 될 수 있을 것이다.

참고문헌

(1) Bathe, K. J. and Wilson, E. L., 1973 "Solution Methods for Eigenvalue Problems in Structural Mechanics," *Int'l J. for Num. Meth. in Engng*, Vol. 6, pp. 213~226.

(2) Lanczos, C., 1950 "An Iteration Method for the Solution of the Eigenvalue Problem of Linear Differential and Integral Operator," *J. of Research of the National Bureau of Standards*, Vol. 45, pp. 255~282.

(3) Bauer, F. L., 1957 "Das Verfahren der Treppen -Iteration und Verwandte Verfahren zur Loesung Algebraischer Eigenwertprobleme," *Zeitschrift fuer Angewandte Mathematik und Physik*, Vol. 8, pp. 214~235.

(4) Jennings, A., 1967 "A Direct Iteration Method of Obtaining Latent Roots and Vectors of a Symmetric Matrix," *Proc. of Cambridge Philosophical Society*, Vol. 63, pp. 755~765.

(5) Rutishauser, H., 1969 "Computational Aspects of F. L. Bauer's Simultaneous Iteration Method," *Numerische Mathematik*, Vol. 13, pp. 4~13.

(6) Yamamoto, Y. and Ohtsubo, H., 1976 "Subspace Iteration Accelerated by Using Chebyshev Polynomials for Eigenvalue Problems with Symmetric Matrices," *Int'l J. Num. Meth. Engng*, Vol. 10, pp. 935~944.

(7) Bathe, K. J. and Ramaswamy, S., 1980 "An

Accelerated Subspace Iteration Method," *Comp. Meth. in Appl. Mech. and Engng*, Vol. 23, pp. 313~331.

(8) Cheu, T. C., Johnson, C. P. and Craig, Jr. R. R., 1987 "Computer Algorithms for Calculating Efficient Initial Vectors for Subspace Iteration Method," *Int'l J. Num. Meth. Engng*, Vol. 24, pp. 1841~1848.

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

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

(11) Ruhe, A., 1979 "Implemetation Aspects of band Lanczos Algorithm for Computation of Eigenvalues of Large Sparse Symmetric Matrices," *Math. of Comput.*, Vol. 33, pp. 680~687.

(12) Parlett, B. N., 1979 "The Lanczos Algorithm with Selective Orthogonalization," *Math. of Comput.*, Vol. 33, pp. 217~238.

(13) Simon, H. D., 1984 "The Lanczos Algorithm with Partial Reorthogonalization," *Math. of Comput.*, Vol. 42, pp. 115~142.

(14) Omid, B. N., Parlett, B. N. and Taylor, R. L., 1983 "Lanczos Versus Subspace Iteration for Solution of Eigenvalue Problems," *Int'l J. Num. Meth. Engng*, Vol. 19, pp. 859~871.

(15) Golub, G. H. and Underwood, R., 1977 "The Block Lanczos Method for Computing Eigenvalues," *In Mathematical Software III* (Ed. by J. Rice), Academic Press, pp. 361~377.

(16) Matties, H. G., 1985 "A Subspace Lanczos Method for the Generalized Symmetric Eigenproblem," *Computers and Structures*, Vol. 21, pp. 319~325.