SCIENCES
SORBONNE
UNIVERSITÉ

Report Project SFPN

PARALLELIZATION OF THE RESOLUTION OF THE SECULAR EQUATION IN OPENMP

May 29, 2019

Students :
BAMBA Ibrahim
NITCHEU Paul

Supervisor :
ABBAS TURKI Lokmane

Contents

Introduction

The symmetric eigenvalue problem is one of the most fundamental problems of computational mathematics. We have many more algorithms available for the symmetric eigenproblem which offer us more flexibility and efficiency. For example, the Bisection algorithm, Tridiagonal QR iteration. Generally for symmetric matrices, we proceed to a tridiagonalization of Householder then to a method of decomposition in eigenelements of the tridiagonal matrix. This offer us one of the fastest methods currently to find all eigenvalues and eigenvectors of symmetric tridiagonal matrices : the Divide-and-conquer method [2]. The basic concept behind these method is the approach from computer science. An eigenvalue problem is divided into two problems of roughly half the size, each of these are solved recursively, and the eigenvalues of the original problem are computed from the results of these smaller problems. This technique provides a fast and accurate alternative to the QR method or to bisection with inverse iteration. The Divide-and-conquer algorithm for solving symmetric tridiagonal eigenproblems was first developed by Cuppen based on previous ideas of Golub and Bunch [6]. A continuing element in divide-and-conquer algorithms descended from Cuppen's [3, 4] is the solution of an eigensystem that differs from diagonal by a rank-1 perturbation. This method has also evolved from work by Dongarra, Sorensen, Tang, and most recently Gu and Eisenstadt [5, 6]. But at the heart of all this methods is the solution of an equation so-called Secular Equation. The term "secular" comes from the latin "saecularis" which is related to "saeculum", which means "century". So secular refers to something that is done or happens every century. It is also used to refer to something that is several centuries old. It appeared in mathematics to denote equations related to the motion of planets and celestial mechanics [7]. The Divide-and-conquer therefore uses the resolution of the secular equation on disjointed intervals simultaneously. Thus, this phase is paralleling well. There are several methods to solve the secular equation. However, the purpose of our work is to adapt a parallel version in OpenMP, on the one hand, two important of them. The first algorithm is monotonous and has a cubic convergence rate [1]. It is called Gragg method. The second algorithm is a hybrid scheme, it is less conventional and combines two methods and is quite effective. On the other hand, it will be a question of comparing these two algorithms and examining the different points of divergence between them. In the rest of this paper, we will first present the Divide-and-conquer method on symmetric tridiagonal matrices showing how it mainly uses the solving of the secular equation. Secondly, we will detail the two secular equation solvers that we are studying before presenting how we implemented them and our choices. Finally, we will show the results of the performances of the two algorithms, their analysis and the result of our comparative study on these algorithms.

I) DIVIDE-AND-CONQUER AND SECULAR EQUATION

In computer science, divide and conquer is an important algorithm design paradigm. This technical approach consists in dividing an initial problem into sub-problems, then solving the sub-problems (recursively or directly if they are small enough) and finally calculating a solution to the initial problem from the solutions of the sub-problems. The strategy becomes :

i. partition the tridiagonal eigenvalue problem into two (or more) smaller

ii. tridiagonal eigenvalue problems.

iii. solve the two smaller problems.

iv. combine the solutions of the smaller problems to get the desired solution of the overall problem.

Evidently, this strategy can be applied recursively. The problem we consider is the following : given a real $n * n$ symmetric matrix T, find all of the eigenvalues and corresponding eigenvectors of T.

Let

$$T = \left[\begin{array}{cccc|cccc}
a_1 & b_1 & & & & & & \\
b_1 & \ddots & \ddots & & & & & \\
 & \ddots & a_{m-1} & b_{m-1} & & & & \\
 & & b_{m-1} & a_m & b_m & & & \\
\hline
 & & & b_m & a_{m+1} & b_{m+1} & & \\
 & & & & b_{m+1} & \ddots & & \\
 & & & & & & \ddots & b_{n-1} \\
 & & & & & & b_{n-1} & a_n
\end{array}\right]$$

A tridiagonal real symmetric matrix T can be decomposed into the sum of two matrices,

$$= \left[\begin{array}{cccc|cccc} a_1 & b_1 & & & & & & \\ b_1 & \ddots & \ddots & & & & & \\ & \ddots & a_{m-1} & b_{m-1} & & & & \\ & & b_{m-1} & a_m - b_m & & & & \\ \hline & & & & a_{m+1} - b_m & b_{m+1} & & \\ & & & & b_{m+1} & \ddots & & \\ & & & & & & \ddots & b_{n-1} \\ & & & & & & b_{n-1} & a_n \end{array}\right]$$

$$+ \left[\begin{array}{c|c} & \\ b_m & b_m \\ \hline b_m & b_m \\ & \end{array}\right]$$

$$= \left[\begin{array}{c|c} T_1 & 0 \\ \hline 0 & T_2 \end{array}\right] + b_m \cdot \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 1 \\ 0 \\ \vdots \\ 0 \end{bmatrix} [0, \ldots, 0, 1, 1, 0, \ldots, 0] \equiv \left[\begin{array}{c|c} T_1 & 0 \\ \hline 0 & T_2 \end{array}\right] + b_m v v^T.$$

Let $T = Q_i D_i Q_i^T$, for $i = 1, 2$, be the eigen decomposition of T_i, where Q_i and D_i are respectively orthogonal and diagonal. That is, Q_i and D_i form the eigenpair of T_i. Thus we have :

$$\begin{aligned} T &= \begin{bmatrix} T_1 & 0 \\ 0 & T_2 \end{bmatrix} + b_m v v^T \\ &= \begin{bmatrix} Q_1 D_1 Q_1^T & 0 \\ 0 & Q_2 D_2 Q_2^T \end{bmatrix} + b_m v v^T \\ &= \begin{bmatrix} Q_1 & 0 \\ 0 & Q_2 \end{bmatrix} (D + \rho v v^T) \begin{bmatrix} Q_1^T & 0 \\ 0 & Q_2^T \end{bmatrix} \end{aligned}$$

where

$$b_m = \rho, u = \begin{bmatrix} Q_1^T & 0 \\ 0 & Q_2^T \end{bmatrix}, D = \begin{bmatrix} D_1 & \\ & D_2 \end{bmatrix}$$

To find the eigenvalues of $D+\rho v v^T$, assume first that $D-\lambda I$ is nonsingular, and compute the characteristic polynomial as follows :

$$det(D+\rho u u^T - \lambda I) = det((D-\lambda I)(I+\rho(D-\lambda I)^{-}1 u u^T)) \qquad (1)$$

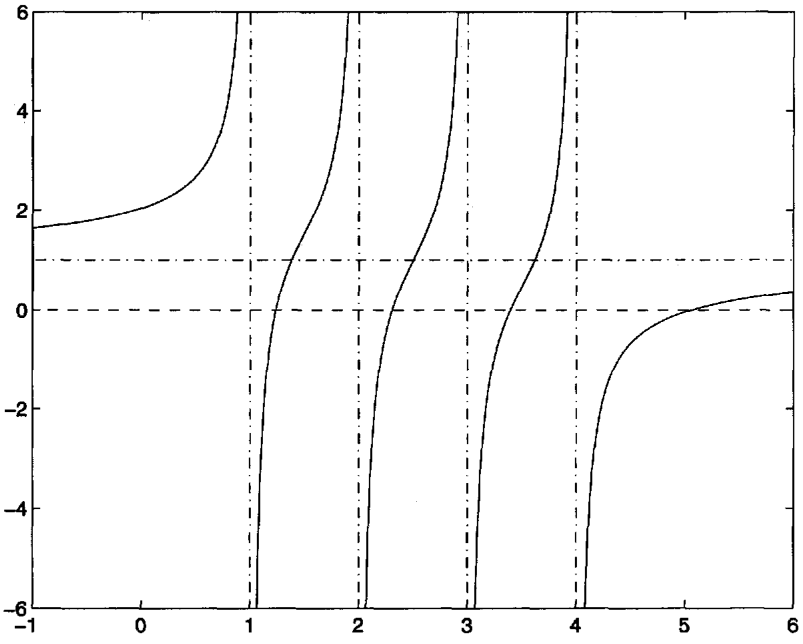
Lemma : If x and y are vectors, $det(I+xy^T) = 1+y^T x$ [2]
Therefore equation (1) becomes :

$$det(I+\rho(D-\lambda I) u u^T) = 1+\rho u^T (D-\lambda I)^{-}1 u$$

$$= 1+\rho\sum_{i=1}^{k}\frac{u_i^2}{d_i-\lambda} \qquad (2)$$

f is called the secular function.And the eigenvalues of T are the roots of the so-called secular equation $f(\lambda)$.So to obtain the eigenvalues of T, we have to solve the secular equation. Let see an example of graph of the secular function (figure 0).

Figure 0: Graph of secular function with 4 intervals



It is easy to see that solving this equation parallels very well at intervals.

II) SECULAR EQUATION SOLVERS

In what follow, to avoid some approximation errors due to the computes of $f(x)$, we divide $f(x)$ to two functions $\psi(x)$ and $\phi(x)$ such that :

$$\psi(x) = \rho \sum_{j=1}^{k} \frac{u_j^2}{d_j - x}, \qquad \phi(x) = \rho \sum_{j=k+1}^{n} \frac{u_j^2}{d_j - x} \tag{3}$$

where $\psi(x)$ correspond to the sum of the negative terms of the secular function and $\phi(x)$ to the sum of the positive terms. We have so :

$$f(x) = 1 + \psi(x) + \phi(x) \tag{4}$$

Then it is easy to see that $f(x)$ has precisely n zeros λ_i, one in each open interval $(d_j; d_{j+1})$, and one to the right of d_n if $\rho > 0$ or to the left of d_1 otherwise. Thus, we set $d_{n+1} = d_n + \sum_{i=1}^{n} \frac{u_i^2}{\rho}$ if $\rho > 0$, and $d_0 = d_1 + \sum_{i=1}^{n} \frac{u_i^2}{\rho}$ otherwise. To simplify our presentation, in the following we assume $\rho > 0$. Then, the n eigenvalues of $\lambda_1,...,\lambda_n$ of the matrix T satisfy

$$d_i < \lambda_i < d_{i+1}, \qquad i = 1,...,n \tag{5}$$

1) Gragg method

This method [1] is based on the a third-order approximation of the form

$$Q(x; c, s, S) = c + \frac{s}{d_k - x} + \frac{S}{d_{k+1} - x} \tag{6}$$

The main problem in finding the eigenvalues by this method is that of providing a stable and efficient method for finding the zeros of secular function. Let the secular function

$$f(\lambda) = 1 + \rho \sum_{i=1}^{n} \frac{u_i^2}{d_i - \lambda}$$

if the d_k are ordered so $d_1 > d_2 > ... > d_{n-1}$, then f has exactly one zero in each of the open intervals $]-\infty, d_{n-1}],]d_{n-1}, d_{n-2}], ..., [d_1, +\infty[$.

The algorithm that we will present in the following is globally convergent in each interval mentioned above except on the last interval. The stop criterion on our monotonous intervals is therefore trivial the problem here will be the last interval where the function

is not monotonous. Thus the criterion of stopping for non-monotonous intervals will be treated in the part **II.4**.
Let y be fixed approximation to λ_k somewhere between $]d_k, d_{k+1}]$ allow us to write

$$f(y) = c + \frac{s}{d_k - y} + \frac{S}{d_{k+1} - y} \tag{7}$$

The idea for computing a correction η to y for the next ("better") approximation $y+\eta$ to λ_k is to solve the equation $Q(x; c, s, S) = 0$ [5]. Gragg proposed to chose c, s, S so that $Q(x; c, s, S)$ matches $f(\lambda)$ at y up to the second derivate. So we have:

$$f'(y) = \frac{s}{(d_k - y)^2} + \frac{S}{(d_{k+1} - y)^2} \tag{8}$$

$$\frac{f''(y)}{2} = \frac{s}{(d_k - y)^3} + \frac{S}{(d_{k+1} - y)^3} \tag{9}$$

with (7), (8), (9) yield:

$$s = \frac{{\Delta_k}^3 \Delta_{k+1}}{\Delta_k - \Delta_{k+1}} \left(\frac{f'(y)}{\Delta_{k+1}} - \frac{f''(y)}{2}\right)$$

$$= u_k^2 + \frac{(d_k - y)^3}{d_k - d_{k+1}} \sum_{i \neq k, k+1}^{n} \frac{d_i - d_{k+1}}{(d_k - y)^3} u_i^2 > u_k^2$$

$$S = \frac{{\Delta_k}^3 \Delta_{k+1}}{\Delta_{k+1} - \Delta_k} \left(\frac{f'(y)}{\Delta_k} - \frac{f''(y)}{2}\right)$$

$$= u_{k+1}^2 + \frac{(d_{k+1} - y)^3}{d_{k+1} - d_k} \sum_{i \neq k, k+1}^{n} \frac{d_i - d_k}{(d_i - y)^3} u_i^2 > u_{k+1}^2$$

$$c = f(y) - (\Delta_k - \Delta_{k+1}) f'(y) + \Delta_k \Delta_{k+1} \frac{f''(y)}{2}$$

with $\Delta_k = d_k - y$.
[5] shows a surprising fact that the iteration formula by solving 6 depends only upon c, alone. As surely, the equations is solvable for any c. Now we calculate next approximation $y_{j+1} = y_j + \eta$ with :

$$\eta = \frac{2b}{a + \sqrt{a^2 - 4bc}} \tag{10}$$

where

$$a = (\Delta_k + \Delta_{k+1}) f(y) - \Delta_k \Delta_{k+1} f'(y), \qquad b = (\Delta_k + \Delta_{k+1}) f(y) \tag{11}$$

So, we have the following algorithm (algorithm 1) to find the approximation λ_k of the zero of $f(x)$ on the interval $]d_k, d_{k+1}[$ by Gragg method.

Algorithm 1 Calculate approximation λ_k of the root of f by Gragg Method

```
Require: $k$
Ensure: $\lambda_k$ between $d_k$ and $d_{k+1}$
  $j \leftarrow 0$
  $\lambda_j \leftarrow initialguess$ on $]d_k, d_{k+1}[$
  repeat
    $a \leftarrow compute\ a$ on $]d_k, d_{k+1}[$
    $b \leftarrow compute\ b$ on $]d_k, d_{k+1}[$
    $c \leftarrow compute\ c$ on $]d_k, d_{k+1}[$
    {Computing the stop criteria}
    if $(\rho > 0$ and $k \neq n-1)$ or $(\rho < 0$ and $k \neq 0)$ then
      $stop \leftarrow (\lambda_{j-1} - \lambda_{j-2}) * (\lambda_{j-1} - \lambda_j)$ {intervals with monotonous}
    else
      $stop \leftarrow compute\ stop\ criteria\ on\ non\ monotonous\ interval$ {interval without monotonous}
    end if
    $\lambda_{j+1} \leftarrow \lambda_j + \eta$ {eta computed by 10
  until stop is false
  return $\lambda_j$
```

The Gragg method is thus easy to implement. It uses only one scheme. However, in addition to the evaluation of $f(x)$ and $f'(x)$, it is called $f"(x)$. When the size of the matrix becomes very large, this evaluation can also have a significant additional cost. Although this does not prevent Gragg's method from remaining fast and efficient thanks to its cubic convergence, it slows it down. It is then necessary to find another more efficient method.

2) Hybrid method

The hybrid method combines two other methods of solving the secular equation that are the middle way method and the fixed weight method by switching between the two when necessary. But in some cases, it uses two poles to speed iterations. Before explaining how the hybrid method works, let's first introduce each of these two methods.

a) Middle way Method

This is a method for solving the secular equation. It will consist of interpolating $\psi(x)$ and $\phi(x)$ by rationals of type $G(x; d, r, s)$ taking both nearby poles into consideration.

To be specific, when $0<k<n$. We Let

$$r+\frac{s}{d_k-x} \text{ an approximation of } \psi(x), and$$
$$R+\frac{S}{d_{k+1}-x} \text{ an approximation of } \phi(x)$$

with $\begin{cases} s=\Delta_k^2\psi'_k(y)>0, \\ r=\psi_k(y)-\Delta_k\psi'_k(y)\leq 0, \end{cases}$ and $\begin{cases} S=\Delta_{k+1}^2\phi'_k(y)>0, \\ R=\phi_k(y)-\Delta_{k+1}\phi'_k(y)\geq 0, \end{cases}$

and $\Delta_k=d_k-y<0<\Delta_{k+1}=d_{k+1}-y$ as before.
Now, we compute our new better approximation $y+\eta$ to λ_k by solving the equation:

$$\rho+\psi(x)+\phi(x) \tag{12}$$

Equation (12) has 2 roots x, one of them infinite if $\rho+r+R=0$. The root we need is the one between d_k and d_{k+1}. Set $\eta=x-y$ Then the correction η to y is:

$$\eta=\frac{a-(-1)^i\sqrt{a^2-4bc}}{2c} \text{ if } a\leq 0$$
$$\frac{2b}{a+(-1)^i\sqrt{a^2-4bc}} \text{ if } a\geq 0$$

With $i=0$ if $k\neq 0$ and 1 otherwise

$$a=(\Delta_k+\Delta k+1)f(y)-\Delta_k\Delta k+1f'(y),$$
$$b=\Delta_k\Delta k+1f(y),$$
$$c=f(y)-\Delta_k\psi'_k(y)-\Delta_{k+1}\phi'_k(y)$$

b) Fixed weight Method

In the middle way method, they exist in cases where the weights u_k or u_{k+1} can be overestimated. It is in order to manage these cases effectively that the fixed weight method has been set up. It will fix one of the weights u_k and u_{k+1} while satisfying (7), (8)

i. The case λ_k closer to d_k: We set $s=u_k^2$ then

$$s=u_k^2, \tag{13}$$

$$S = \Delta_{k+1}^2 (f'(y) - \frac{u_k}{\Delta_k}), \tag{14}$$

$$= u_{k+1}^2 + \textstyle\sum_{j \neq k,k+1} \frac{\Delta k+1^2}{\Delta j^2} u_j^2 > u_{k+1}^2,$$

$$c = f(y) - \frac{u_k^2}{\Delta_k} - \Delta k + 1(f'(y) - \frac{u_k^2}{\Delta k^2}) \tag{15}$$

$$= f(y) - \Delta_{k+1} f'(y) - \tfrac{u_k^2}{\Delta_k^2}(d_k - d_{k+1})$$

ii. The case λ_{k+1} closer to d_k : we set $S = u_{k+1}^2$

$$s = \Delta_k^2 (f'(y) - \frac{u_{k+1}}{\Delta_{k+1}}), \tag{16}$$

$$= u_k^2 + \textstyle\sum_{j \neq k,k+1} \frac{\Delta k^2}{\Delta j^2} u_j^2 > u_k^2,$$

$$S = u_{k+1}^2, \tag{17}$$

$$c = f(y) - f'(y)\Delta_k - \frac{u_{k+1}^2}{\Delta k + 1^2}(d_{k+1} - d_k) \tag{18}$$

To finish this algorithm, we just interpolate $Q(x, c, s, S)$ and find next approximation y $+$ η to λ_k

c) Hybrid scheme

The Hybrid Method is a combination of the two previous methods. To make the iterations faster, it uses in some cases three poles : d_{k-1}, d_k and d_{k+1} instead of d_k and d_{k+1}. For this, it uses the following function :

$$f_m(x) = \rho + \sum_{j=1, j \neq m}^{n} \frac{u_j^2}{d_j - x} \tag{19}$$

which is the secular function $f(x)$ with the mth term in the summation removed. It is easy to see that $f_m(x)$ has a zero between d_{m-1} and d_{m+1} [5].
The Hybrid Method consists in evaluating the secular function $f(x)$ on the interval with the method in a) or b) according to certain conditions that we will present later. But, there are some possible difficult cases one [5] :
<u>case 1</u> : $d_k < \lambda_k < \frac{d_k+d_{k+1}}{2}$ and $f_k(x)$ has a zero between d_k and λ_k
<u>case 2</u> : $\frac{d_k+d_{k+1}}{2} < \lambda_k < d_{k+1}$ and $f_{k+1}(x)$ has a zero between λ_k and d_{k+1}.
To see why the case 1 may be difficult, we let $u_k^2 \to 0$, then, as functions of u_k^2, λ_{k-1} goes monotonically upward until it hits d_k while λ_k goes monotonically downward until it hits the zero of $f_k(x)$ between d_k and λ_k. Now, on the contrary, let u_k^2 go back from zero to its original value, what we will see is the exact contrary phenomena. Based on this simple observation, we can think, roughly, that λ_{k-1} depends largely on the pole d_k and its weight u_k^2 and, therefore, the Fixed Weight Method should be good at finding it. On the other hand, λ_k depends on the pole d_k and its weight u_k^2 and the zero of $f_k(x)$ where it starts from and which is controlled roughly by the the poles d_{k-1} and d_{k+1} with appropriate weights. Similar intuitive arguments apply to the second case above [5]. If we are in case 1, the way to handle it is to interpolate $f(x)$ with :

$$\tilde{Q}(x;c,s,S) =^{def} c + \frac{s}{d_{k-1}-x} + \frac{u_k^2}{d_k-x} + \frac{S}{d_{k+1}-x} \tag{20}$$

where parameters c, s and S can be determined by either interpolating $f_k(x) = \rho+\psi_{k-1}(x)+\phi_k(x)$ with the Fixed Weight Method or the Middle Way Method depending on situation. If we are in case 2, we interpolate $f(x)$ with (20) where c, s and S are determined by interpolating $f_{k+1}(x) = \rho + \psi_k(x) + \phi_{k+1}(x)$ with the Fixed Weight or the Middle Way Method. Once we have c, s and S, the zero between d_k and d_{k+1} is calculated by iterative methods with negligible costs. We use here the dichotomy between d_k and $\frac{d_k+d_{k+1}}{2}$ if we are in case 1, $\frac{d_k+d_{k+1}}{2}$ and d_{k+1} if we are in case 2. Since we have approximated $f(x)$ with $\tilde{Q}$, to evaluate f, we have to evaluate $\tilde{Q}$. However, care has to be taken in evaluating $\tilde{Q}(x;c,s,S)$ at a given point. As a matter of fact, because of roundoff, sometimes(though rarely) computed $\tilde{Q}(y;c,s,S)$ differs significantly from computed $f(y)$ though the two should be the same by interpolation in theory. It turns out that we can evaluate $\tilde{Q}(x;c,s,S)$ in an indirectly way to avoid this from happening. To avoid calculation costs, since computed $f(y)$ is available at the time we interpolate the secular function $f(x)$ (see algorithm), we do not compute $\tilde{Q}(y;c,s,S)$ at all while simply setting it to be $f(y)$. At the very next time when we need to compute $\tilde{Q}(x;c,s,S)$ we update $f(y)$ by adding a correction to it as following [5] :

$$\begin{aligned}\tilde{Q}(x;c,s,S) &= \tilde{Q}(y;c,s,S) + (\tilde{Q}(x;c,s,S) - \tilde{Q}(y;c,s,S)) \\ &= f(y) + (x-y)\left(\frac{s}{(d_{k-1}-y)-(x-y)} \times + \frac{u_k^2}{(d_k-y)-(x-y)} + \frac{S}{(d_{k+1}-x)-(x-y)}\right)\end{aligned} \tag{21}$$

Because subsequent evaluation of $\tilde{Q}(x;c,s,S)$ is done in the same way by adding correction to its value at the previous x.

As mentioned before, the hybrid method uses three poles in some cases to accelerate iterations. Suppose that we are computing λ_k and we have an initial guess $d_k+\tau$ for λ_k for which $d_k < d_k+\tau < \lambda_k$. The value of the secular function can be evaluated by :

$$f(d_k+\tau) = f_k(d_k+\tau) + \frac{u_k^2}{-\tau} \tag{22}$$

So, we can obtain $f_k(d_k+\tau)$ as a by-product

$$f_k(d_k+\tau) = f(d_k+\tau) - \frac{u_k^2}{-\tau} \tag{23}$$

Thus,

if $f_k(d_k+\tau>0)$, **then**

two poles d_k **and** d_{k+1} **are used;**

else

we are in one of the difficult cases, three poles d_{k-1}, d_k **and** d_{k+1} **are used;**

The hybrid scheme combines the Middle Way and the Fixed Weight by switching between both. For the first iteration, we use the Fixed Weight Method to interpolate $f(x)$ or $f_k(x)$ according the situation because it's know that, in average, it's faster than Middle Way Method. We get so a new approximation $d_k+\tau_1$ of λ_k. For the second iteration, we switch to the Middle Way Method when $f(d_k+\tau_1)<0$ and $|f(d_k+\tau_1)| > 0.1 * |f(d_k+\tau)|$. For the next iterations, we switch from one method to another when $f_{new} * f_{pre} > 0$ and $|f_{new}| > 0.1 * |f_{pre}|$ where f_{new} is the value of the secular function at the newest approximation $d_k+\tau_i$ and f_{pre} at the previous one $(d_k+\tau_{i-1})$.

We obtain so the following algorithm which calculates the approximation λ_k of the zero of the secular function on interval $]d_k, d_{k+1}[$ (*see algorithm 2*).

Algorithm 2 Calculate approximation λ_k of the root of f by Hybrid Scheme

```
Require: $k$
Ensure: $\lambda_k$ between $d_k$ and $d_{k+1}$
  $j \leftarrow 0$
  $y_j \leftarrow initialguess$ on $]d_k, d_{k+1}[$
  compute $f_k(y_j)$
  {Taking decision}
  if $f_k(y_j) > 0$ then
    two poles $d_k$ and $d_{k+1}$ are used
    we interpolate $f(x)$ with Fixed Weight
  else
    three poles $d_{k-1}$, $d_k$ and $d_{k+1}$ are used
    if we are in the case 1 then
      we interpolate $f_k(x)$ as in (20) with Fixed Weight
    end if
    if we are in the case 2 then
      we interpolate $f_{k+1}(x)$ as in (20) with Fixed Weight
    end if
  end if
  $j \leftarrow j+1$
  if $f(y_j) < 0$ and $|f(y_j)| > 0.1 * |f(y_{j-1})|$ then
    restart the first iteration and use Middle Way
  else
    restart the first iteration and continue to use Fixed Weight
  end if
  $j \leftarrow j+1$
  while stop criteria is false do
    if $f(y_j) * f(y_{j-1}) > 0$ and $|f(y_j| > 0.1 * |f(y_{j-1})|$ then
      switch method
    end if
    restart iteration with the method chosen
    $j \leftarrow j+1$
  end while
  return $y_j$
```

3) Initial Guess

We have seen that, whether in the Gragg Method or the Hybrid Method, the iterations begin with the choice of an initial approximate value for λ_k called the *initial guess*. One of the major difficulties in implementing the algorithms described above is the choice of this initial guess. Indeed, it may happen that an iteration that should converge quadratically converges slowly or not at all because we took a bad initial guess. Moreover, it is also important to know which of d_k and d_{k+1} is closes to λ_k. Indeed, if decided wrongly, round-off could cause an iterate to collide with an endpoint d_k or d_{k+1}, or even overshoot it and this collision would soon lead to division by Zero. In what follows, we will present an inexpensive method [5] to obtain a relatively accurate value of initial guess. At the same time we shall show how to decide which of d_k and d_{k+1} is closes to λ_k.

i. The case $1 < k < n$

We rewrite the secular function as $f(x) = g(x) + h(x)$ where,

$$g(x) = \rho + \sum_{j=1, j\neq k,k+1} \frac{u_j^2}{d_j - x} \quad and \quad h(x) = \frac{u_k^2}{d_k - x} + \frac{u_{k+1}^2}{d_{k+1} - x} \tag{24}$$

We choose our initial guess y to be that one of the two roots of the equation

$$g(\frac{d_k + d_{k+1}}{2}) + h(y) = 0 \tag{25}$$

In case $f(\frac{d_k+d_{k+1}}{2}) \geq 0$, equation (25) should be solved for $\tau = y - d_k$ and while in case $f(\frac{d_k+d_{k+1}}{2}) < 0$, it should be solve for $\tau = y - d_{k+1}$. Define $\Delta = d_k - d_{k+1}$ and $c = g(\frac{d_k-d_{k+1}}{2})$.The τ formulas are:

$$\tau = y - d_k = \frac{a - \sqrt{a^2 - 4bc}}{2c} \quad if\, a \leq 0, \tag{26}$$

$$= \frac{2b}{a + \sqrt{a^2 - 4bc}} \quad if\, a \leq 0$$

where if $f(\frac{d_k+d_{k+1}}{2}) \geq 0$

$$K = k, \; a = c\Delta + (u_k^2 + u_{k+1}^2), \; b = u_k^2 \Delta, \tag{27}$$

and if $f(\frac{d_k+d_{k+1}}{2}) < 0$

$$K = k+1,\ a = -c\Delta + (u_k^2 + u_{k+1}^2),\ b = -u_{k+1}^2\Delta \tag{28}$$

with theses formulas, the algorithm is completed to find τ in $1 < k < n$ Always in this same interval,we will show with how to decide which of d_k and d_{k+1} is closes for λ_k.

Theorem: if $f(\frac{d_k+d_{k+1}}{2}) > 0$, then τ given by (26) and (27) satisfies

$$d_k < d_k + \tau < \lambda_k < \frac{d_k + d_{k+1}}{2}$$

if $f(\frac{d_k+d_{k+1}}{2}) < 0$, then τ given by (26) and (28) satisfies

$$\frac{d_k + d_{k+1}}{2} < \lambda_k < d_{k+1} + \tau < d_{k+1}$$

Proof: for 1<k<n , we have,

$$-g(\lambda_k) = \frac{u_k^2}{d_k - \lambda_k} + \frac{u_{k+1}^2}{d_{k+1} - \lambda_k}$$

$$-g(\frac{d_k + d_{k+1}}{2}) = \frac{u_k^2}{d_k - y} + \frac{u_{k+1}^2}{d_{k+1} - y}$$

which produces

$$(\lambda_k - y)(\frac{u_k^2}{(d_k - \lambda_k)(d_k - y)} + \frac{u_{k+1}^2}{(d_{k+1} - \lambda_k)(d_{k+1} - y)})$$

$$= (\frac{d_k + d_{k+1}}{2} - \lambda_k) \sum_{j \neq k,k+1} \frac{u_j^2}{(d_j - \frac{d_k - d_{k+1}}{2})(d_j - \lambda_k)}$$

Thus $\lambda_k - y$ has the same sign as $\frac{d_k+d_{k+1}}{2} - \lambda_k$

ii. The case $k = n$

We partition the secular function as $f(x) = g(x) + h(x)$ and we get the initial guess y by solving:

$$g(\frac{d_n + d_{n+1}}{2}) + h(y) = 0$$

Here are the formulas for τ :
ii-1-The case $\frac{d_n+d_{n+1}}{2} < \lambda_n$,i.e,$f(\frac{d_n+d_{n+1}}{2}) \leq n$:

a. if $g(\frac{d_n+d_{n+1}}{2}) \leq -h(d_{n+1})$ then $\tau = y - d_n = \frac{zz^T}{\rho}$

b. if $g(\frac{d_n+d_{n+1}}{2}) > -h(d_{n+1})$ then

$$\tau = y - d_k = \frac{a + \sqrt{a^2 - 4bc}}{2c} \quad if a \leq 0, \tag{29}$$

$$= \frac{2b}{a - \sqrt{a^2 - 4bc}} \quad if a \leq 0$$

where $\Delta = d_n - d_{n-1}$, $c = g(\frac{d_n+d_{n+1}}{2})$ and

$$a = -c\Delta + (u_n^2 + u_{n-1}^2), \; b = u_n^2\Delta, \tag{30}$$

It can be proved that in this case $\frac{d_n+d_{n+1}}{2} \leq \lambda_n < d_n + \tau < d_{n+1}$

ii-2-The case $\frac{d_n+d_{n+1}}{2} < \lambda_n$,i.e,$f(\frac{d_n+d_{n+1}}{2}) > 0$.Then $g(\frac{d_n+d_{n+1}}{2}) > -h(d_{n+1})$.
We compute $\tau = y - d_n$ by (29) and (30).It is easy to show that in this case $d_n < d_n + \tau < \lambda_n < \frac{d_n+d_{n+1}}{2}$

4) Stopping Criteria

Since the hybrid algorithm is not monotonous like that of gragg, it is necessary to implement a good stopping criterion so that we do not have an infinite loop. Ren-Cang Li[5] will propose two criteria that can solve this problem.To guarantee full accuracy of computed distance the stopping criterion was set to be:

$$|\eta| \leq c\epsilon_m min(|d_k - x|, |d_{k+1} - x|)$$

where η is the last iterative correction that was computed,c is fairly small constant, ϵ_m is the machine's roundoff threshold and , x is the current iterate.

However, the criteria for stopping will be considered

$$\eta^2 \leq \epsilon_m(\eta - \eta_0) min(|d_k - x|, |d_{k+1} - x|)$$

where η_0 is the correction before last.This new stopping criterion often save one iteration per eigenvalue found.

Gu and Eisenstadt [8] found a new way to compute eigenvectors after all eigenvalues are computed.They proposed a stop criterion which is expressed as follows:

$$|f(x)| \leq n\epsilon_m(\rho + \sum_{j=1}^{n} |\frac{u_j^2}{d_j - x}|) \tag{31}$$

But the problem with this stopping criterion is that for a large n, this can allow an excessive error, whereas for a small n, the risk is that it is never satisfied.Their implementation is as follows: it calculates $\psi_k(x)$ by summing each term of j = 1 to k, while $\phi_k(x)$ from j = n to k + 1, because hoping in this way to add the terms from small magnitude to large. Note that we are actually working with

$$f(d_K + \tau) = \rho + \sum_{j=1}^{n} \frac{u_j^2}{(d_j - d_K) - \tau} \tag{32}$$

where $\tau = x - d_k$ and $K = k$ if λ_k comes more close to d_k than to other d_j and $K = k+1$ otherwise. It is easy to show that

$$|f(d_K + \tau) - (Computed f(d_K + \tau)| \leq \epsilon_m e,$$

with $e = 2\rho + \sum_{j=1}^{k} (k - j + 6)|\frac{u_j^2}{d_j - x}| + \sum_{j=n}^{k+1} (j - k + 5)|\frac{u_j^2}{d_j - x}| + |f(d_K + \tau)|$

which can be computed recursively on our way to compute $\psi_k(x)$ and $\phi_k(x)$ and costs about $2n$ additions for each iteration cycle. On the other hand, if τ is a neighbor of the desired zero τ^* to the function (32) i.e $|\tau - \tau^*| \leq \epsilon_m|\tau|$ then

$|f(d_k + \tau) - 0| \leq |\tau - \tau^*||f'(d_k + \tau)| + O(|\tau - \tau^*|^2)$
$\leq |\tau - \tau^*||f'(d_k + \tau)| + O(|\tau - \tau^*|^2)$

$$f(d_k + \tau) - 0| \leq |\tau - \tau^*||f'(d_k + \tau)| + O(|\tau - \tau^*|^2) \tag{33}$$

$$\leq |\tau - \tau^*||f'(d_k + \tau)| + O(|\tau - \tau^*|^2)$$

In view of this, the stop criterion is defined as

$$|f(d_k + \tau)| \leq e\epsilon_m + \epsilon_m|\tau||f'(d_k - \tau)| \quad (34)$$

III) PERFORMANCE TESTS, RESULTS AND ANALYSIS

In this section, we present the results of the performance measures performed on each method and the error of our eigenvalues compared to the evaluation of $f(x)$. Note that all our tests were performed on the machine **ppti-14-509-01** (of Sorbonne University, campus Jussieu), an *Intel-core i7* 8-core machine.

Figure 1. Execution time/matrice length with 2, 4 and 8 threads

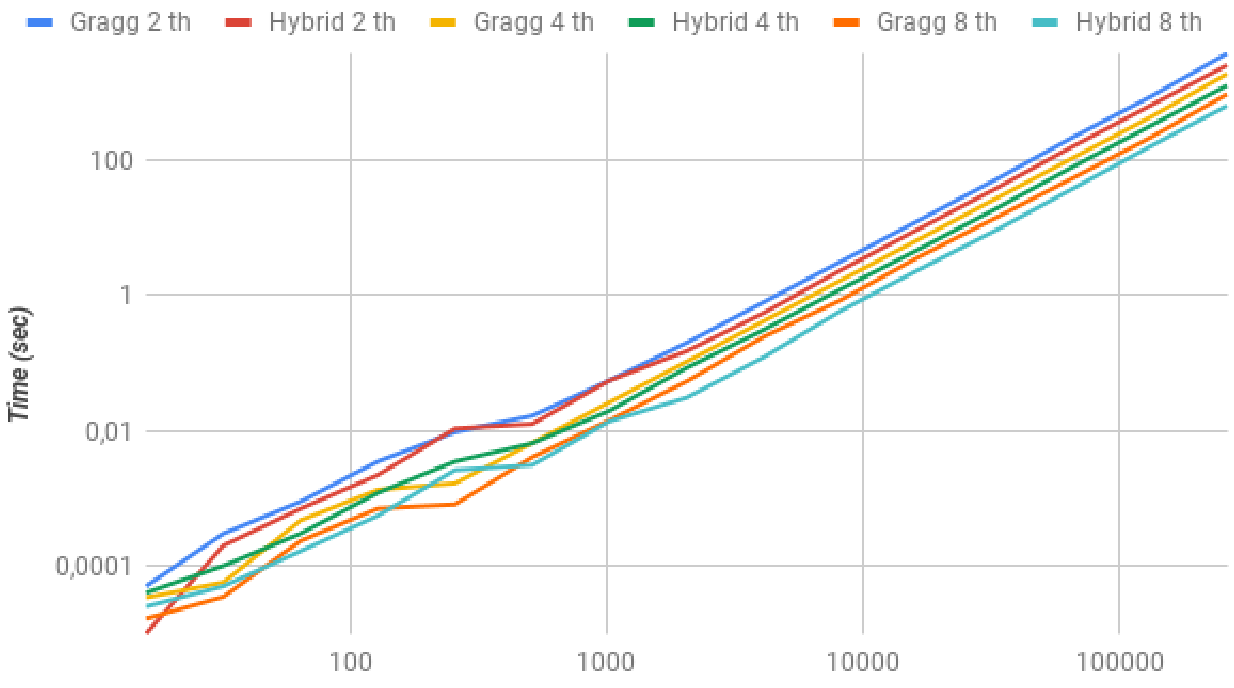


Figure 1 shows the average execution time curves of the Gragg and Hybrid algorithms compared to the matrix size for 2, 4 and 8 threads. This figure shows that, for the two algorithms, the larger the size of the matrix, the more the execution time is important.

Indeed, when the size of the matrix is multiplied by 2, the execution time is multiplied by approximately 4 (see table 1). That's why we have quasi-linear curves. We also observe that the increase in the number of threads brings a real gain in execution time for the two algorithms. In fact, the resolution of the secular equation is easily parallelizable by interval, the search for solution being independent on each of them. Moreover, at equal number of threads, the Hybrid Method is always faster than Gragg Method and the time difference becomes more and more significant when the size of the matrix is large. This could be explained by the fact that in the Gragg algorithm, besides the evaluation of the secular function $f(x)$ and its first derivative $f'(x)$, its second derivative $f"(x)$ is evaluated. Thus, when the size of the matrix becomes very large, this evaluation can have a very important additional cost as shown in *Figure 1*. Finally, it is interesting to note that for matrices of small size ($n < 1000$), the parallelization does not always bring a gain in performance, on the contrary, we see a loss of performance. This is due to the cost of communication between threads. Indeed, we have a dynamic parallelization in order to balance the computing load between the threads. However, when the size of the matrix is very small, it is preferable to carry out a static parallelization to avoid the cost of the communications. This remark is more meaningful in the case of matrices smaller than 10. We see that the Hybrid Method with 2 threads for example is more efficient than those with 4 and 8 threads. Increasing the number of threads increases the communication time between threads which is greater than the computation time.

Figure 2. Speed-up compared to number of threads

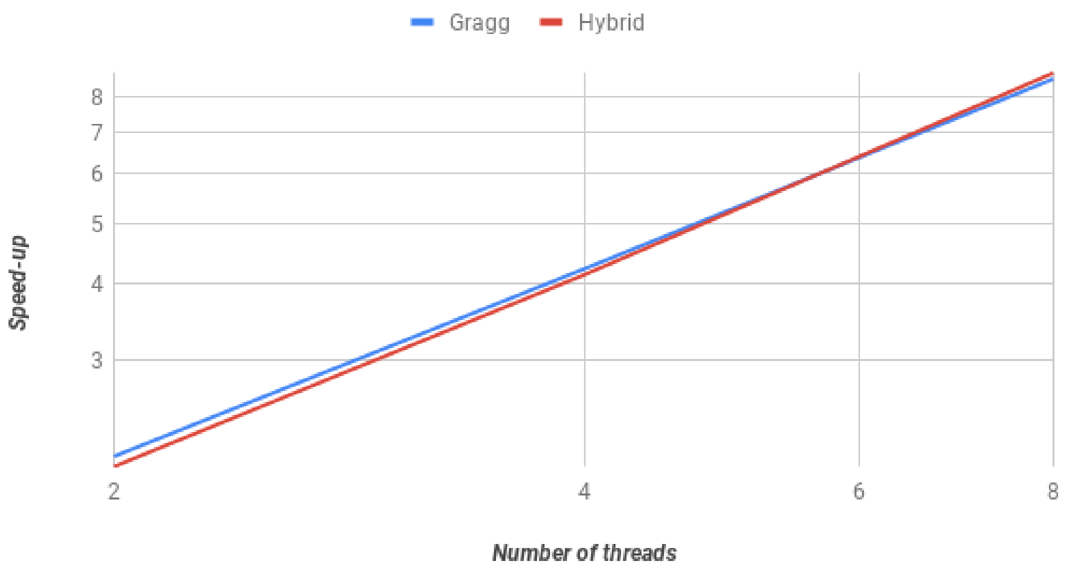
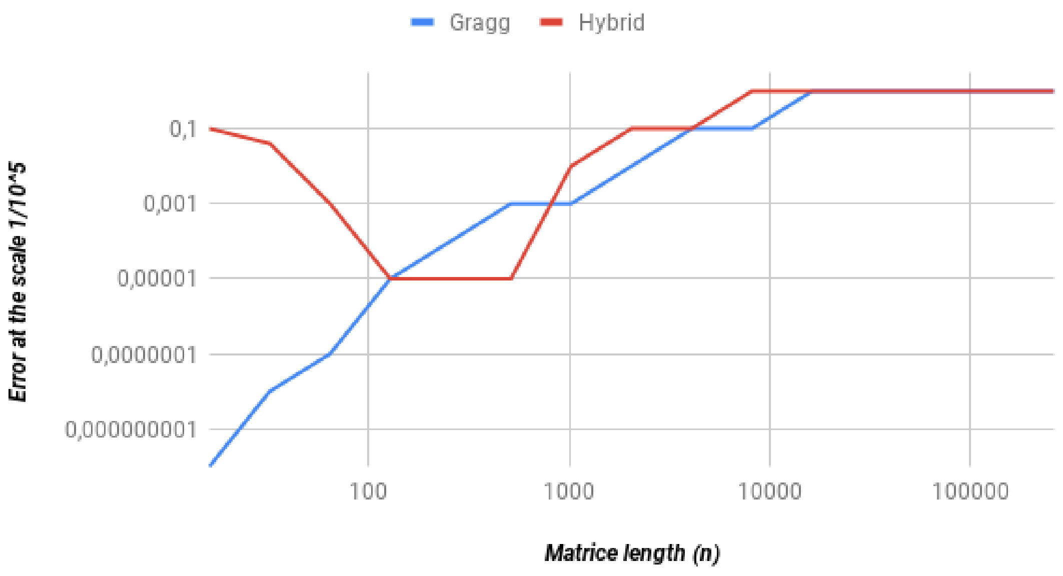


Figure 2 shows the speed-up compared to the number of processes for the two algorithms. Note that the speed-up is defined as the execution time of the sequential algorithm divided by the time of the parallel algorithm. Both curves are linear. This implies that when we double the number of threads for a given matrix size, its execution time is halved.

Figure 3. Error in evaluation of f



In order to measure the precision of the eigenvalue approximation on an interval for each of the two algorithms(Gragg Method and Hybrid Scheme) and to know how far our eigenvalues are from true eigenvalues, we have evaluated, for each eigenvalue λ_k produced by our algorithm, $f(\lambda_k)$ and then measure its distance to zero. *Figure 3* presents the results obtained, the curve of the error of the evaluation of f compared to the size of the matrix.

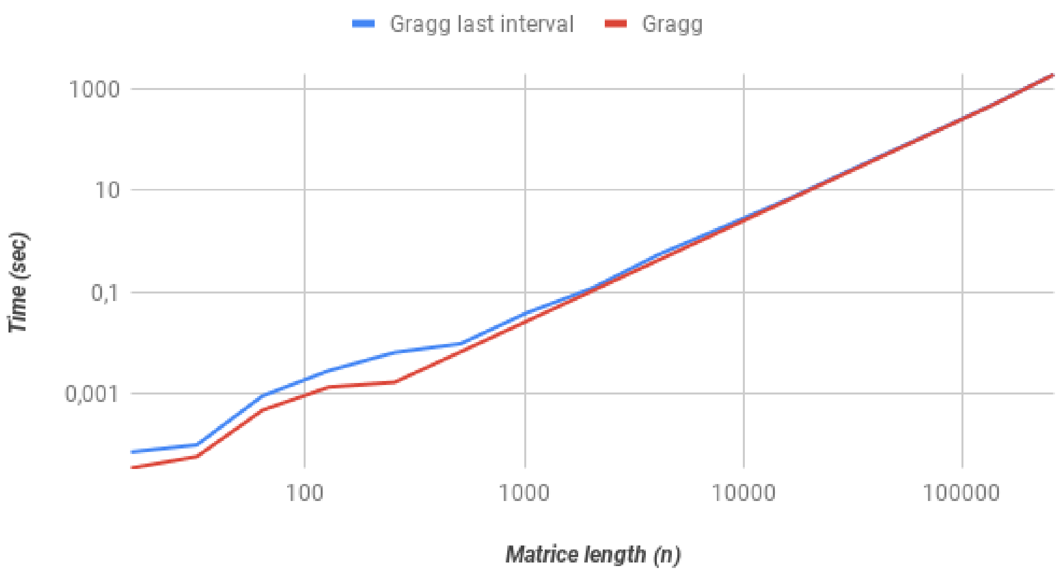
The curve of the Gragg error (blue curve) shows that, for matrices of reasonable sizes, this method is extremely precise (of the order of 10^{-15} to 10^{-9}). But, it is less and less precise when the size of the matrix increases, the error increases with the increase of size until reaching 10^{-5} (when the size of the matrix is higher than 16000). Value with which it remains constant regardless of the following increases in size.

For the curve of the error of the Hybrid Method (red curve), it shows that the Hybrid method is not very precise in the set. For matrices of very small size, the precision is very low (of the order of 10^{-5}). However, it increases for medium-sized matrices until it reaches its best value (10^{-10} for a size matrix of about 125). Then, precision decreases with increasing error as the size of the matrix increases (> 500) to its worst value (10^{-5}), at which point it remains constant.

Thus, from the two curves, there is a large difference in precision between the Gragg Method and that of Hybrid for a certain size of matrix. Indeed, during the algorithm of the Hybrid Method, we make an approximation of the eigenvalue with an inexpensive iterative method (in our case, we use the dichotomy) which leads to a loss of precision while the algorithm of Gragg proceeds to the evaluation of the second derivative of the secular function for a better approximation.

In view of the above, it can be deduced that the Gragg Method, although less fast, has a better precision than Hybrid even if for some matrix size, in some cases, the Hybrid Method has a better precision. However, in these cases, the difference between Hybrid and Gragg is very small (of the order of 10^{-1}). Moreover, when the size of the matrix becomes very large (> 16000), the two methods have a constant and equal precision. Thus, in these cases, the hybrid method will be favored given its speed.

Figure 4. Gragg last interval compared to the average of intervals



To finish this section of tests and analysis, we come back for a little on Gragg's Method. Indeed, this method has a cubic convergence on the intervals from the diagonal of the tridiagonal matrix. It is therefore monotonous on these intervals which is however not the case on the last added interval. Gragg Method loses its monotony on this interval which make slow the execution on this interval. While studying the execution time by measuring the average of the execution time of all the intervals is certainly preferable (as in *Figure 1*), it would also be beneficial to study the same time by measuring the time of the slower interval (the last interval). *Figure 4* shows the execution time of the Gragg method when the time is the average of the times of all the intervals (*curve red*), and the time of the last interval (*curve blue*). It shows that the run time on the added interval is all the time more important than the average time. We can deduce that this interval decreases the performance of the Gragg algorithm. Also, we can think that the Gragg method is slow because of this interval.

Conclusion

Throughout this work, we have studied two methods of interpolation of secular equation. Each with different strengths that we have highlighted and verified through several tests. A combination of the appropriate fixed weight and Middle way method leads to the hybrid scheme that competes with the Gragg cubic convergent scheme on random problems. However, as our numerical results show, the hybrid method has a much better execution time than Gragg (about 1.5 times less).But it is important to specify, that in the algorithm Gragg the time of the computation on the last interval is slowed the computation of all the others.Finally, in this work ti is important to remember that for the resolution of the secular equation,the hybrid method is better than Gragg when one is more concerned about the execution time. accuracy of the eigenvalues.We are interested in, the hybrid method is worse than Gragg method.

References

[1] William B. Gragg, J. R. Thornton, and D. D. Warner (1992) : Parallel divide and conquer algorithms for the symmetric tridiagonal eigenproblem and bidiagonal singular value problem, *Modeling and Simulation*, 23(1), 49-56, Clemson University

[2] James Demel (1997) : Apply numerical linear algebra, *SIAM*, 229-233

[3] J. J. M. Cuppen (1981) : A divide and conquer method for the symmetric tridiagonal eigenproblem, *Numer. Math.*, 36, 177-195

[4] J. J. Dongarra and D. C. Sorensen (1987) : A fully parallel algorithm for the symmetric eigenvalue problem, *SIAM J. Sci. Stat. Comput.*, 8, s139-s154

[5] Ren-Cang Li (1993) : Solving Secular Equations Stably and Efficiently, *Computer Science Dept. Technical Report CS-94-260*, Department of Mathematics, University of California at Berkeley, Berkeley, California 94720

[6] G. H. Golub (1973) : Some modified matrix eigenvalue problems, *SIAM rev.*, 15, 318-334

[7] Gerard Meurand (2012) : Solving secular equations, 3-7

[8] Ming Gu and S. C. Eisenstadt (1992) : A stable and efficient algorithm for the rank one modification of the symmetric eigenproblems, *Research Report YaleU/DCS/RR 916*, Yale University