Parallel solving for the secular equation using OpenMP

Basile Arlandis Basile Lewandowski

conducted under the supervision of Pr. Lokmane ABBAS TURKI

Previous work [2, 5] has shown that the symmetric eigenvalue problem is solvable efficiently using the *Secular equation*, thanks to a *Divide and Conquer* algorithm. This process also has the benefit of being highly parallelizable. We propose here an overview of the available algorithms and their implementation using OpenMP.

Keywords: *Eigendecomposition, Secular equation, Divide and Conquer, OpenMP*

Contents

Introduction

The eigendecomposition of a real symmetric matrix is a classical numerical algebra problem. To proceed, we first use Householder's tridiagonalization and then apply a divide and conquer scheme to solve the secular equation. The next goal is to find the roots of the so-called secular function through an interpolation algorithm. Compared to classical *Singular Value Decomposition* (whose complexity is cubic in this case [6]) this process is of quadratic complexity.
We will here consider two schemes converging towards the eigenvalues, and eventually compare their performances.

1 Secular Equation solving

To begin, we first proceed to a tridiagonalization. We consider a real symmetric matrix S, let T be the tridiagonal matrix given by its Householder decomposition, as detailed in [7]. The point is here that S and T have the same eigenvalues while the tridiagonalization is more time-efficient than a full diagonalization.

Lemma 1.1 (Householder Lemma) *For all real symmetric matrix S, there is an orthogonal matrix H so that $H^\top SH$ is symmetric tridiagonal.*

Divide phase

$$T = \begin{pmatrix} \alpha_1 & \beta_1 & & & & \\ \beta_1 & \alpha_2 & \beta_2 & & & \\ & \beta_2 & \ddots & \ddots & & \\ & & \ddots & \ddots & \ddots & \\ & & & \ddots & \ddots & \beta_{n-1} \\ & & & & \beta_{n-1} & \alpha_n \end{pmatrix}$$

Once the matrix is of tridiagonal shape, it can be divided into two smaller problems as follow :

$$
T = \left(\begin{array}{ccccc|ccccc}
\alpha_1 & \beta_1 & & & & & & & & \\
\beta_1 & \ddots & \ddots & & & & & & & \\
 & \ddots & \ddots & \beta_{m-1} & & & & & & \\
 & & \beta_{m-1} & \alpha_m & \beta_m & & & & & \\
\hline
 & & & \beta_m & \alpha_{m+1} & \beta_{m+1} & & & & \\
 & & & & \beta_{m+1} & \ddots & \ddots & & & \\
 & & & & & \ddots & \ddots & \beta_{n-1} & & \\
 & & & & & & \beta_{n-1} & \alpha_n & &
\end{array}\right)
$$

$$
= \left(\begin{array}{ccccc|ccccc}
\alpha_1 & \beta_1 & & & & & & & & \\
\beta_1 & \ddots & \ddots & & & & & & & \\
 & \ddots & \ddots & \beta_{m-1} & & & & 0 & & \\
 & & \beta_{m-1} & \alpha_m - \beta_m & & & & & & \\
\hline
 & & & & & \alpha_{m+1} - \beta_m & \beta_{m+1} & & & \\
 & & 0 & & & \beta_{m+1} & \ddots & \ddots & & \\
 & & & & & & \ddots & \ddots & \beta_{n-1} & \\
 & & & & & & & \beta_{n-1} & \alpha_n &
\end{array}\right)
+ \left(\begin{array}{ccccc|ccccc}
 & & & & & & & & & \\
 & & & & & & & & & \\
 & & & & & & & & & \\
 & & & & & \beta_m & \beta_m & & & \\
\hline
 & & & & & \beta_m & \beta_m & & & \\
 & & & & & & & & & \\
 & & & & & & & & & \\
 & & & & & & & & &
\end{array}\right)
$$

$$
= \left(\begin{array}{c|c} T_1 & \\ \hline & T_2 \end{array}\right) + \beta_m \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \\ 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} (0 \dots 0\ 1\ 1\ 0 \dots 0) := \left(\begin{array}{c|c} T_1 & \\ \hline & T_2 \end{array}\right) + \beta_m v v^\top
$$

Conquer phase

We now suppose we have the eigendecompostitions of both submatrices, so that $T_i = Q_i D_i Q_i^\top$ with D_i diagonal and Q_i orthogonal. We now have :

$$\begin{aligned} T &= \begin{pmatrix} T_1 & 0 \\ 0 & T_2 \end{pmatrix} + \beta_m v v^\top \\ &= \begin{pmatrix} Q_1 D_1 Q_1^\top & 0 \\ 0 & Q_2 D_2 Q_2^\top \end{pmatrix} + \beta_m v v^\top \\ &= \begin{pmatrix} Q_1 & 0 \\ 0 & Q_2 \end{pmatrix} \left(\begin{pmatrix} D_1 & \\ & D_2 \end{pmatrix} + \rho^{-1} \xi \xi^\top \right) \begin{pmatrix} Q_1^\top & 0 \\ 0 & Q_2^\top \end{pmatrix} \end{aligned}$$

as we denote $\left(\begin{array}{c|c} Q_1^\top & \\ \hline & Q_2^\top \end{array}\right)$ by ξ and $\rho = \frac{1}{\beta_m}$.
Considering we have :

$$\det(T) = \det(D + \rho^{-1} \xi \xi^\top)$$

We proceed with the characteristic polynomial as follow :

$$\begin{aligned} \det(T - \lambda I) &= \det(D + \rho^{-1} \xi \xi^\top - \lambda I) \\ &= \det((D - \lambda I)(I + \frac{1}{\rho}(D—\lambda I)^{-1} \xi \xi^\top) \end{aligned}$$

Assuming $D - \lambda I$ is non-singular, the determinant vanishes wherever λ is an eignenvalue.

Lemma 1.2 *Considering two vectors x and y,* $\det(I + xy^\top) = 1 + y^\top x$

Hence, we obtain :

$$\begin{aligned} \det((D - \lambda I)(I + \frac{1}{\rho}(D—\lambda I)^{-1} \xi \xi^\top) &= 1 + \frac{1}{\rho} \xi^\top (D—\lambda I)^{-1} \xi \\ &= \rho + \sum_{i=1}^{n} \frac{\xi_i^2}{d_i - \lambda} \\ &:= f(\lambda) \end{aligned}$$

We have thus defined the *secular function* f so that every root of f is an eigenvalue of T. If all d_i are distinct and all $u_i \neq 0$, the function f has the graph shown below :

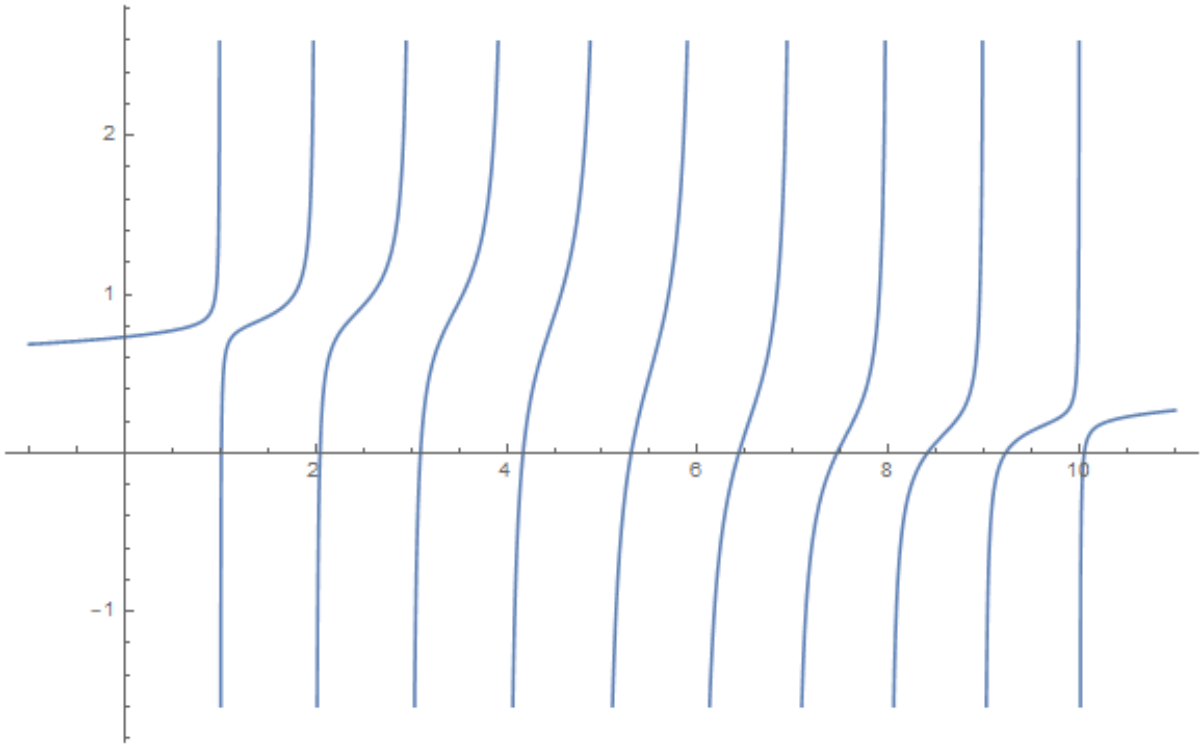

Figure 1. Secular function for a rank 10 matrix

As we can see, the line $y = \rho$ is a horizontal asymptote, and the lines $\lambda = d_i$ are vertical asymptotes. It can be proven that we can then compute the eigenvector by $(D - \lambda I)^{-1}$ with $O(n)$ flops (cf. [4]). Further explanation and proof can be found in [3, 9].
Once this function is properly defined, we can find the eigenvalues thanks to a root-finding algorithm. We will here introduce two different algorithms to approximate these roots.

2 Root Finding

In order to compute the secular function f at a decent speed, we approximate this function with terms as simple as possible.
A first idea is to divide the second term of the secular function

$$\sum_{i=1}^{n} \frac{\zeta_i^2}{d_i - \lambda}$$

in two terms $\psi_k(x)$ and $\phi_k(x)$ such as :

$$\psi_k(x) = \sum_{i=1}^{k} \frac{\zeta_i^2}{d_i - \lambda}$$
$$\phi_k(x) = \sum_{i=k+1}^{n} \frac{\zeta_i^2}{d_i - \lambda}$$
$$f(x) = \rho + \psi_k(x) + \phi_k(x)$$

k depending upon which eigenvalue λ_k is being computed. If this equation approximates f really precisely, it is still too slow to compute f values, thus we choose to interpolate $f(x)$ with the equation $f(x) = c + \frac{s}{d_k - x} + \frac{S}{d_{k+1} - x}$, where $\psi_k(x) = r + \frac{s}{d_k - x}$, $\phi_k(x) = R + \frac{S}{d_{k+1} - x}$, and $c = \rho + r + R$.

We base our algorithms on the interpolation of $f(x)$ at y by :

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

for which

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

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

r, R, s and S are defined in [4] by :

$$s = \Delta_k^2 \psi_k'(y) > 0$$
$$S = \Delta_{k+1}^2 \phi_k'(y) > 0$$
$$r = \psi_k(y) - \Delta_k \psi_k'(y) \leq 0$$
$$R = \phi_k(y) - \Delta_{k+1} \phi_k'(y) \geq 0$$

where $\Delta_k = \delta_k - y$ and $\Delta_{k+1} = \delta_{k+1} - y$.

The philosophy behind our root finding is to compute $f(\lambda_k)$ with $y = \lambda_k$ using an iteration formula as long as the stopping criterion is not reached, and apply a correction η to y for the next better approximation $y + \eta$.

In practice, we do not use both s and S in our pseudo-code and calculate η using the formulas given in [5]:

$$\begin{aligned}\eta &= \frac{a - \sqrt{a^2 - 4bc}}{2c} \quad \text{if} \quad a \leq 0 \\ &= \frac{2b}{a + \sqrt{a^2 - 4bc}} \quad \text{if} \quad a > 0\end{aligned}$$

where

$$\begin{aligned}a &= (\Delta_k + \Delta_{k+1})f(y) - \Delta_k \Delta_{k+1} f'(y) \\ b &= \Delta_k \Delta_{k+1} f(y).\end{aligned}$$

At this point, we compute c in a different way in each algorithm, we will here give more explanation about this computation.

Gragg's Scheme

In Gragg's scheme, we compute c using f second derivative, such as :

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

In order to be correct, this equation requires that $Q(x; c, s, S)$ matches $f(x)$ up to the second derivative, thus it implies :

$$\begin{aligned}s &= \frac{\Delta_k^3 \Delta_{k+1}}{\Delta_k - \Delta_{k+1}} \left(\frac{f'(y)}{\Delta_{k+1}} - \frac{f''(y)}{2}\right) \\ &= \zeta_k^2 + \frac{(\delta_k - y)^3}{\delta_k - \delta_{k+1}} \sum_{i \neq k, k+1} \frac{\delta_i - \delta_{k+1}}{(\delta_i - y)^3} \zeta_i^2 \;>\; \zeta_k^2 \\ S &= \frac{\Delta_k \Delta_{k+1}^3}{\Delta_{k+1} - \Delta_k} \left(\frac{f'(y)}{\Delta_k} - \frac{f''(y)}{2}\right) \\ &= \zeta_{k+1}^2 + \frac{(\delta_{k+1} - y)^3}{\delta_{k+1} - \delta_k} \sum_{i \neq k, k+1} \frac{\delta_i - \delta_k}{(\delta_i - y)^3} \zeta_i^2 \;>\; \zeta_{k+1}^2\end{aligned}$$

This algorithm requires more work on the secular function $f(x)$ because it needs to compute f second derivative, but converges monotonically to the desired eigenvalue λ_k, except in the case $k = n$ where the monotonic convergence is lost.

Furthermore, its implementation is really simple as shown in the algorithm thereafter.

Algorithm 1 Calculate approximation λ_k using Gragg's Scheme

Require: k

Ensure: λ_k *between* δ_k *and* δ_{k+1}

$a \leftarrow 0$

$b \leftarrow 0$

$c \leftarrow 0$

$\eta \leftarrow 0$

$y \leftarrow initial\_guess(k)$ see 2.3.1

repeat

$a \leftarrow$ *compute* a *knowing* k

$b \leftarrow$ *compute* b *knowing* k

$c \leftarrow$ *compute* c *knowing* k *and using Gragg's Scheme*

$\eta \leftarrow$ *compute* η *considering* $a \leq 0$ *or* $a > 0$

$y \leftarrow y + \eta$

until $|f(y)| > e\epsilon_m + \epsilon_m|y||f'(2\delta_k - y)|$ see 2.3.2

return y

In order to the algorithm to work for $k = n$, we approximate δ_{k+1} by

$$\delta_{k+1} = \sum_{i=1}^{n} \frac{\zeta_i^2}{\rho}$$

as described in [5]. This guarantees us that $\delta_n < \lambda_n < \delta_{n+1}$ and at the same time keeps a decent computing time for approximating λ_n.

But, when the matrices length are increasing, computing $f''(y)$ slows the algorithm down. To improve our time performances, we can use the Hybrid's Scheme.

Hybrid Scheme

The Hybrid Scheme is a smart combination of two methods, *The Middle Way Method* and *The Fixed Weight Method*, that uses the fastest method for a given k and switches to the

other one during the algorithm if the computation of λ_k is slowing down.
Both *The Middle Way Method* and *The Fixed Weight Method* compute a better approximation $y+\eta$ to λ_k by solving the equation $Q(x;c,s,S) = c + \frac{s}{d_k - x} + \frac{S}{d_{k+1} - x}$ (1) described above and satisfy (2) and (3). However, they differ in computing c as we are about to see.

The Middle Way Method

The Middle Way Method takes into consideration both poles δ_k and δ_{k+1} to compute c using the equation :

$$\begin{aligned} c &= f(y) - \Delta_k \psi'(y) - \Delta_{k+1}\phi'(y) \\ &= f(y) - \Delta_{k+1} f'(y) - \psi'(y)(\delta_k - \delta_{k+1}) \text{ (described in [4])} \end{aligned}$$

Taking into consideration both poles δ_k and δ_{k+1} gives *The Middle Way Method* a quadratic convergence, but it has some defaults too. Even if most of the time, $r + \frac{s}{d_k - x}$ approximates $\psi(k)$ very well, they are cases when s (respectively S) overestimates ζ_k^2 (respectively ζ_{k+1}^2) so much that the iterations are forced to move slowly towards the desired root. When such a situation occurs, we switch to *The Fixed Weight Method*.

The Fixed Weight Method

In order to prevent over estimations, *The Fixed Weight Method* fixes one of the weights ζ_k^2 or ζ_{k+1}^2 such as :

1. when λ_k is closer to δ_k :

$$\begin{aligned} s &= \zeta_k^2 \\ S &= \Delta_{k+1}(f'(y) - \frac{\zeta_k^2}{\Delta_k^2}) \\ &= \zeta_{k+1}^2 + \sum_{j \neq k,k+1} \frac{\Delta_{k+1}^2}{\Delta_j^2}\zeta_j^2 \ > \ \zeta_{k+1}^2 \\ c &= f(y) - \frac{\zeta_k^2}{\Delta_k} - \Delta_{k+1}(f'(y) - \frac{\zeta_k^2}{\Delta_k^2}) \\ &= f(y) - \Delta_{k+1}f'(y) - \frac{\zeta_k^2}{\Delta_k^2}(\delta_k - \delta_{k+1}) \end{aligned}$$

2. when λ_k is closer to δ_{k+1} :

$$\begin{aligned} s &= \Delta_k(f'(y) - \frac{\zeta_{k+1}^2}{\Delta_{k+1}^2}) \\ &= \zeta_k^2 + \sum_{j \neq k,k+1} \frac{\Delta_k^2}{\Delta_j^2}\zeta_j^2 \ > \ \zeta_k^2 \\ S &= \zeta_{k+1}^2 \\ c &= f(y) - \frac{\zeta_k^2}{\Delta_k} - \Delta_{k+1}(f'(y) - \frac{\zeta_k^2}{\Delta_k^2}) \\ &= f(y) - \Delta_k f'(y) - \frac{\zeta_{k+1}^2}{\Delta_{k+1}^2}(\delta_{k+1} - \delta_k) \end{aligned}$$

Fixing s or S prevents *The Fixed Weight Method* from overestimating ζ_k^2 or ζ_{k+1}^2, but in the meantime can make iterations converge slower.

Using 2 or 3 poles and switching between methods

To prevent iterations from going slow, *The Hybrid's Scheme* computes $\lambda_k - \delta_k$ instead of λ_k. For a given initial guess $\delta_k + \tau$ for λ_k, the secular function is evaluated by :

$$f(\delta_k + \tau) = f_k(\delta_k + \tau) + \frac{\zeta_k^2}{-\tau} \ < \ 0$$

where $f_k(\delta_k + \tau)$ is obtained as a by-product.
We now decide between using two or three poles : if $f_k(\delta_k + \tau) > 0$ then we use 2 poles δ_k and

δ_{k+1}, else we use 3 poles δ_{k-1}, δ_k and δ_{k+1}.
We now proceed to the first iteration by using *The Fixed Weight Method* to interpolate $f(x)$ if the decision favors two poles, or $f_k(x)$ if the decision favors three poles and get a new approximation τ_1. Before the second iteration, we switch to *The Middle Way Method* if $f(\delta_k + \tau_1) < 0$ and $|f(\delta_k + \tau_1)| > 0.1 * f(\delta_k + \tau)$.
We can now run the algorithm properly and we switch from one method to another as soon as iterations are slowing down. The criterion that indicates iterations are slowing down is given by:

$$f_{new} f_{pre} > 0 \ \ \& \ \ |f_{new}| > 0.1 * |f_{pre}|$$

where f_{pre} and f_{new} are the values of the secular function respectively at the previous and the new approximation.

Algorithm 2 Calculate approximation λ_k using The Hybrid's Scheme

Require: k

Ensure: λ_k *between* δ_k *and* δ_{k+1}

$a \leftarrow$ *compute* a *knowing* k

$b \leftarrow$ *compute* b *knowing* k

$y \leftarrow initial\_guess(k)$ see 2.3.1

$f_{pre} \leftarrow f(y)$

$f_k(y) \leftarrow f(y) - \frac{\zeta_k^2}{\lambda_k - \delta_k}$

$switch \leftarrow true$

$k_j \leftarrow k$

if $f_k(y) > 0$ **then**

$c \leftarrow$ *compute* c *knowing* k *and using* $f(y)$ *and The Fixed Weight Method*

else

$c \leftarrow$ *compute* c *knowing* k *and using* $f_k(y)$ *and The Fixed Weight Method*

end if

$\eta \leftarrow$ *compute* η *considering* $a \leq 0$ *or* $a > 0$

$y \leftarrow y + \eta$

$f_{new} \leftarrow f(y)$

if $f_{new} < 0$ & $|f_{new}| > 0.1 * |f_{pre}|$ **then**

$switch \leftarrow false$

end if

if $|y - \delta_k| > |y - \delta_{k+1}|$ **then**

$k_j \leftarrow k + 1$

end if

repeat

$a \leftarrow$ *compute* a *knowing* k

$b \leftarrow$ *compute* b *knowing* k

$c \leftarrow$ *compute* c *knowing* k *and using Middle Way or Fixed Weight Method depending on switch*

$\eta \leftarrow$ *compute* η *considering* $a \leq 0$ *or* $a > 0$

$y \leftarrow y + \eta$

$f_{pre} \leftarrow f_{new}$

$f_{new} \leftarrow f(y)$

until $|f(y)| > e\epsilon_m + \epsilon_m |y - \delta_{kj}||f'(2\delta_{kj} - y)|$ see 2.3.2

return y

Initial guess and Stopping Criteria

Initial Guess

To calculate our initial guess, we first look if λ_k is closer to δ_k or δ_{k+1}, in order to prevent an iterate to collide with an endpoint δ_k or δ_{k+1} and therefore cause a *Division by zero.* In order to do so, we naturally look at the sign of $f(\frac{\delta_k+\delta_{k+1}}{2})$ and rewrite the secular function as $f(x) = g(x) + h(x)$ where:

$$g(x) = \rho + \sum_{j=1, j\neq k,k+1}^{n} \frac{\zeta_j^2}{\delta_j - x}$$
$$h(x) = \frac{\zeta_k^2}{\delta_k - x} + \frac{\zeta_{k+1}^2}{\delta_{k+1} - x}$$

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

$$g(\frac{\delta_k + \delta_{k+1}}{2}) + h(y) = 0 \tag{4}$$

If $f(\frac{\delta_k+\delta_{k+1}}{2}) \geq 0$, the equation (4) should be solved for $\tau = y - \delta_k$, else it should be solved for $\tau = y - \delta_{k+1}$. Therefore, our initial guess τ is calculated threw the equation :

$$\begin{aligned} \tau = y - \delta_k &= \frac{a - \sqrt{a^2 - 4bc}}{2c} \quad \text{if} \quad a \leq 0 \\ &= \frac{2b}{a + \sqrt{a^2 - 4bc}} \quad \text{if} \quad a > 0 \end{aligned}$$

where $\Delta = \delta_{k+1} - \delta_k$ and $c = g(\frac{\delta_k+\delta_{k+1}}{2})$, and if $f(\frac{\delta_k+\delta_{k+1}}{2}) \geq 0$:

$$K = k,\ a = c\Delta + (\zeta_k^2 + \zeta_{k+1}^2),\ b = \zeta_k^2\Delta$$

or finally if $f(\frac{\delta_k+\delta_{k+1}}{2}) < 0$:

$$K = k+1,\ a = -c\Delta + (\zeta_k^2 + \zeta_{k+1}^2),\ b = -\zeta_{k+1}^2\Delta$$

. For the case $k = n$, we compute our initial guess using quite the same method except when $g(\frac{\delta_n+\delta_{n+1}}{2}) \leq -h(\delta_{n+1})$ (details are given in [5]).
If $g(\frac{\delta_n+\delta_{n+1}}{2}) > -h(\delta_{n+1})$, then :

$$\begin{aligned} \tau = y - \delta_n &= \frac{a - \sqrt{a^2 - 4bc}}{2c} \quad \text{if} \quad a \leq 0 \\ &= \frac{2b}{a + \sqrt{a^2 - 4bc}} \quad \text{if} \quad a > 0 \end{aligned}$$

where

$$\Delta = \delta_n - \delta_{n-1},\ c = g(\frac{\delta_n + \delta_{n+1}}{2}),\ a = -c\Delta + (\zeta_{n-1}^2 + \zeta_n^2),\ b = \zeta_n^2\Delta$$

and in the case $g(\frac{\delta_n+\delta_{n+1}}{2}) \leq -h(\delta_{n+1})$:

$$\tau = y - \delta_n = \sum_{i=1}^{n} \frac{\zeta_i^2}{\rho}.$$

Stopping Criteria

It has been proved that, in order to guarantee the computed eigenvectors to be fully orthogonal, one must be able to compute the distances between each λ_i and δ_j to almost full accuracy. This double precision was invented to evaluate the secular function extra precisely when necessary and therefore we use the stopping criteria given in [5] with a little modification.
According to *Ibid.*, a stopping criteria giving this needed double precision is given by:

$$|f(\delta_K + \tau)| \leq e\epsilon_m + \epsilon_m|\tau||f'(\delta_K - \tau)|$$

where

$$K = k \text{ if } \lambda_k \text{ comes more close to } \delta_k \text{ than to other } \delta_j \text{ and } K = k+1 \text{ otherwise}$$

$$e = 2\rho + \sum_{j=1}^{k}(k-j+6)|\frac{\zeta_j^2}{\delta_j - x}| + \sum_{j=k+1}^{n}(j-k+5)|\frac{\zeta_j^2}{\delta_j - x}| + |f(\delta_K + \tau)|$$

and ϵ_m is the machine's roundoff threshold (in our case 10^{-16}).
But in practice, using our machine's roundoff threshold made the stopping criterion more accurate than necessary and even caused failures of the Hybrid's Scheme (see 3). In order to prevent this failures from happening, we set $e = 10^{-14}$ in our stopping criteria, which led to both satisfying precision and failure proportion.

3 Benchmark

We ran a series of tests to highlight the differences between these two algorithms. We also evaluated the reliability of the hybrid method in various context.

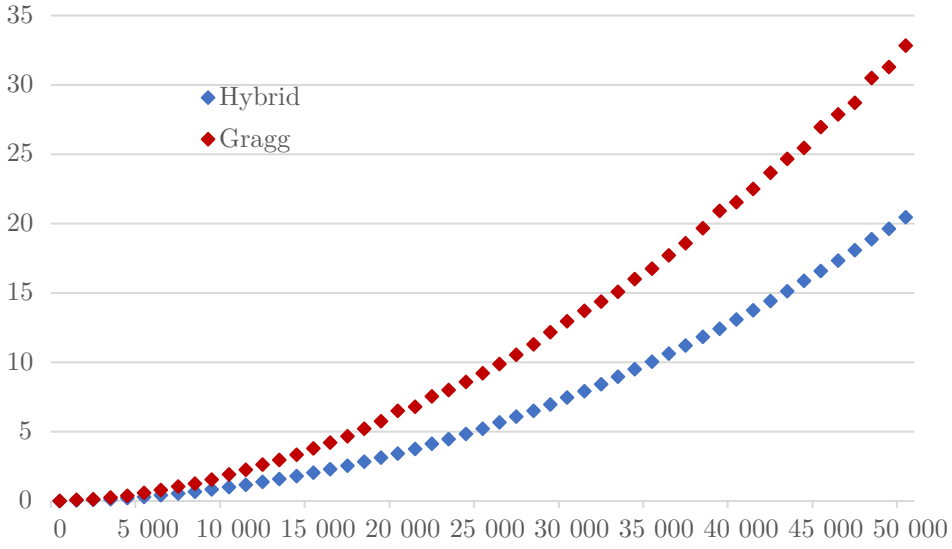

Figure 2. Time efficiency of both algorithms

Computation time in seconds versus matrix size. Program was run on two Intel Xeon E5-2630 v3 (32 threads @ 2.40GHz)

As expected, the *Hybrid* algorithm converge faster than Gragg's method. In both cases, computations are completed within a reasonable duration.

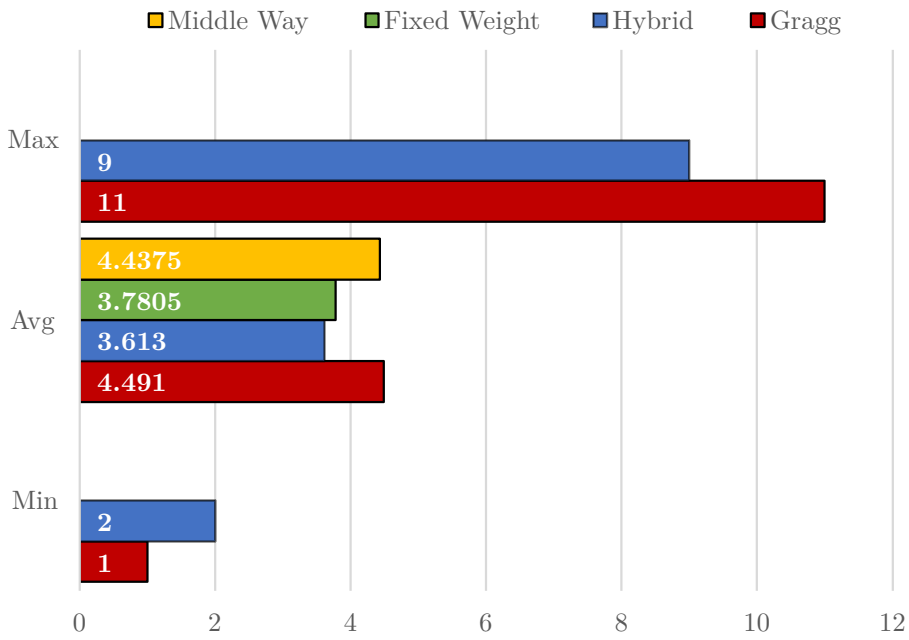

Figure 3. Number of iterations for each eigenvalue
Test on a size 2000 matrix with double precision stopping criterion.

The *Hybrid* algorithm is thus effective on average and in worst cases. Best cases implies at least two iterations because we consider there is an iteration when choosing the amount of poles to use (second iteration occurs before checking the stopping criterion). We could avoid these two iterations in best cases scenarios by checking if the initial guess fulfills the stopping criterion.

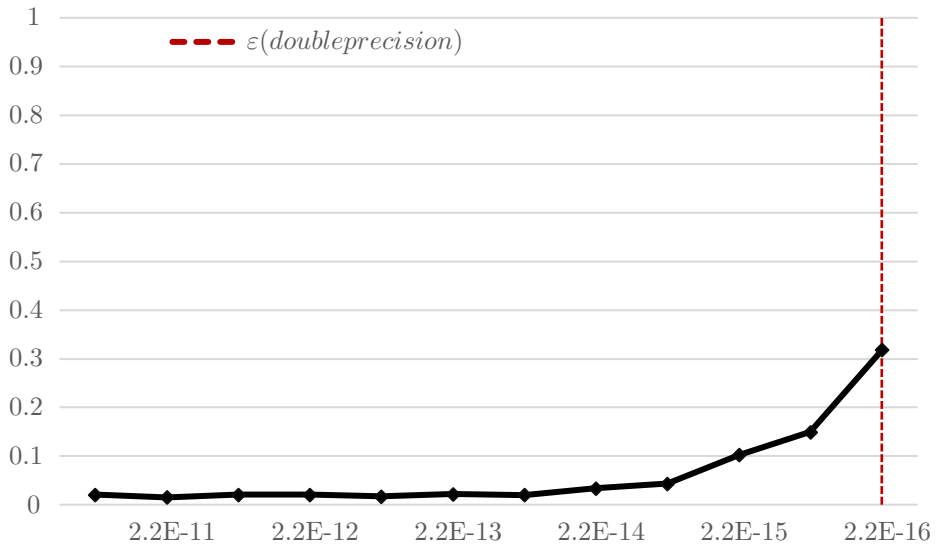

Figure 4. Failure proportion for the Hybrid scheme

Test on 3500 eigenvalues with several stopping criteria. Failure occurs when the algorithm can't meet the precision expected.

The program can fail to return an eigenvalue when the stopping criterion is unreachable. This is typically the case when the eigenvalues are too close to one another or when the increment is too small compared to the machine's roundoff threshold. According to the tests we have run, we reach this limit as the precision used in the stopping criterion come closer to the machine's epsilon ε. A workaround proposed in [5] is to finish by a Newton's iteration in this case, and thus reach the eigenvalue at last.

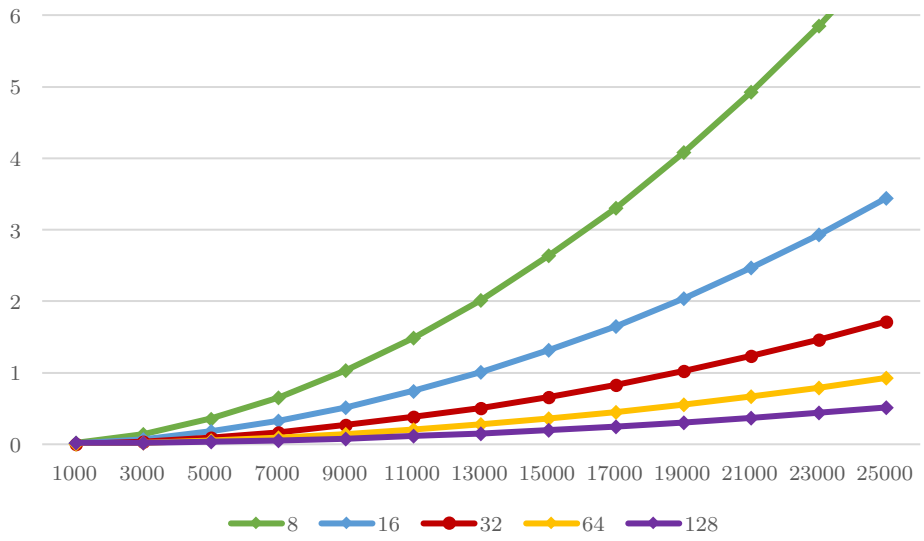

Figure 5. Performance by thread

Computation time in seconds versus matrix size, with different amounts of threads available. Program was run on two AMD EPYC 7452 32-Core Processor

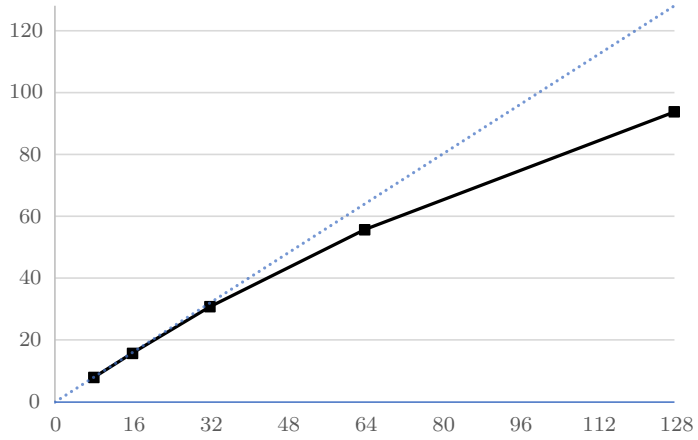

Figure 6. Speedup

Concerning the performances, scaling the thread amount is very effective, and we can notice that the speedup is quasi-linear. On our test, the results appear to be somewhat weaker

on highly-parallelized programs, but we have to keep in mind that computation time is tiny at this scale (less than half a second for a size 20 000 matrix), and might induce some interference.

Conclusion

We have here considered two algorithms to resolve a symmetric eigenvalue problem. Both are efficient to solve this problem, and perform well on multi-threaded systems. Furthermore, we have ensured that Gragg's method was less time efficient than the *Hybrid method*
Further comparison and other algorithms can be found in [8].
The promising results we have obtained with a multi-threaded program suggest an approach using general-purpose processing on gpu could bring more improvement to the performances of these algorithms, as discussed in [1].

Acknowledgments

Experiments were carried out using the Grid'5000 infrastructure. Source code can be found at `github.com/BasileLewan/ParallelizeSecularEquation`.

References

[1] L. A. Abbas-Turki and S. Graillat, *Resolving small random symmetric linear systems on graphics processing units*, The Journal of Supercomputing, 73 (2017), pp. 1360–1386.

[2] C. F. Borges, W. B. Gragg, J. R. Thornton, and D. D. Warner, *Parallel Divide and Conquer Algorithms for the Symmetric Tridiagonal Eigenproblem and Bidiagonal Singular Value Problem*, (1993), pp. 49–56. Accepted: 2014-03-13T15:28:42Z Institution: Monterey, California: Naval Postgraduate School.

[3] J. J. M. Cuppen, *A divide and conquer method for the symmetric tridiagonal eigenproblem*, Numerische Mathematik, 36 (1980), pp. 177–195.

[4] J. W. Demmel, *Applied Numerical Linear Algebra*, Society for Industrial and Applied Mathematics, Jan. 1997.

[5] R.-C. Li, *Solving Secular Equations Stably and Efficiently:*, tech. rep., Defense Technical Information Center, Fort Belvoir, VA, Apr. 1993.

[6] F. Liang, R. Shi, and Q. Mo, *A split-and-merge approach for singular value decomposition of large-scale matrices*, Statistics and Its Interface, 9 (2016), pp. 453–459.

[7] R. S. Martin, C. Reinsch, and J. H. Wilkinson, *Householder's Tridiagonalization of a Symmetric Matrix*, in Linear Algebra, F. L. Bauer, ed., Springer Berlin Heidelberg, Berlin, Heidelberg, 1971, pp. 212–226.

[8] A. Melman, *A numerical comparison of methods for solving secular equations*, Journal of Computational and Applied Mathematics, 86 (1997), pp. 237–249.

[9] A. R. Mitchell, *J. H. Wilkinson, The Algebraic Eigenvalue Problem (Clarendon Press, Oxford, 1965), 662pp., 110s.*, Proceedings of the Edinburgh Mathematical Society, 15 (1967), pp. 328–328.