# EfficientMaxEigenpair Vignette

*Mu-Fa Chen*
*Assisted by Xiao-Jun Mao*

*2016-12-06*

`EfficientMaxEigenpair` is a package for computing the maximal eigenpair of the matrices with non-negative off-diagonal elements (or as a dual, the minimal eigenvalue of the matrices with non-positive off-diagonal elements). This vignette is a simple guide to using the package. All the algorithms and examples provided in this vignette are available in the paper "Efficient initials for computing maximal eigenpair" by Mu-Fa Chen. The paper Chen (2016) is now included in the Vol 4, in the middle of the website:

http://math0.bnu.edu.cn/~chenmf/

Let us install and require the package `EfficientMaxEigenpair` first.

```
require(EfficientMaxEigenpair)
```

Before moving to the details, let us illustrate the algorithm by two typical examples.

## The tridiagonal case (Example 7 in Chen (2016))

The matrices we considered for the tridiagonal case are in the form of

$$Q = \begin{bmatrix} -1 & 1^2 & 0 & 0 & \dots & 0 \\ 1^2 & -5 & 2^2 & 0 & \dots & 0 \\ 0 & 2^2 & -13 & 3^2 & \dots & 0 \\ \vdots & \ddots & \ddots & \ddots & \vdots & \vdots \\ 0 & \dots & \dots & (n-2)^2 & -(n-2)^2-(n-1)^2 & (n-1)^2 \\ 0 & \dots & \dots & 0 & (n-1)^2 & -(n-1)^2-n^2 \end{bmatrix},$$

where $n$ is the dimension of the matrix. For the infinite case, we have known that the maximal eigenvalue is $-1/4$. We now want to calulate the maximal eigenvalues of matrices in different dimensions by the Rayleigh quotient iteration.

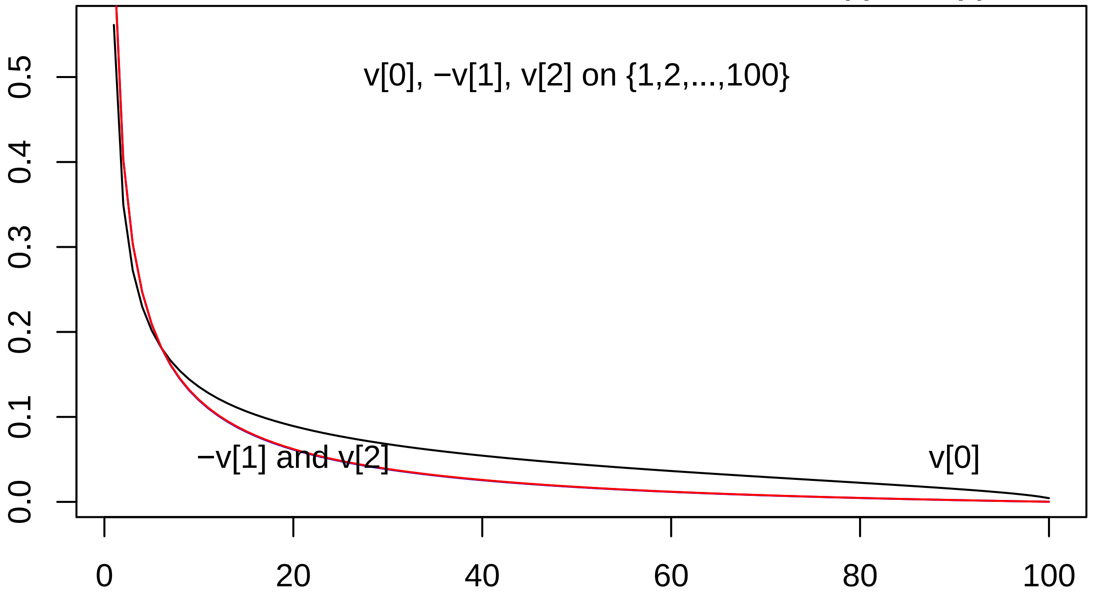
```
# Define the dimension of matrix to be 100.
nn = 100

# Generate the corresponding tridiagonal matrix.
a = c(1:(nn - 1))^2
b = c(1:(nn - 1))^2
c = rep(0, length(a) + 1)
c[length(a) + 1] = nn^2

# Output the corresponding convergence maximal eigenpairs of the tridiagonal
# matrix.
eigenpair = eff.ini.maxeig.tri(a, b, c, xi = 7/8, improved = T)
```

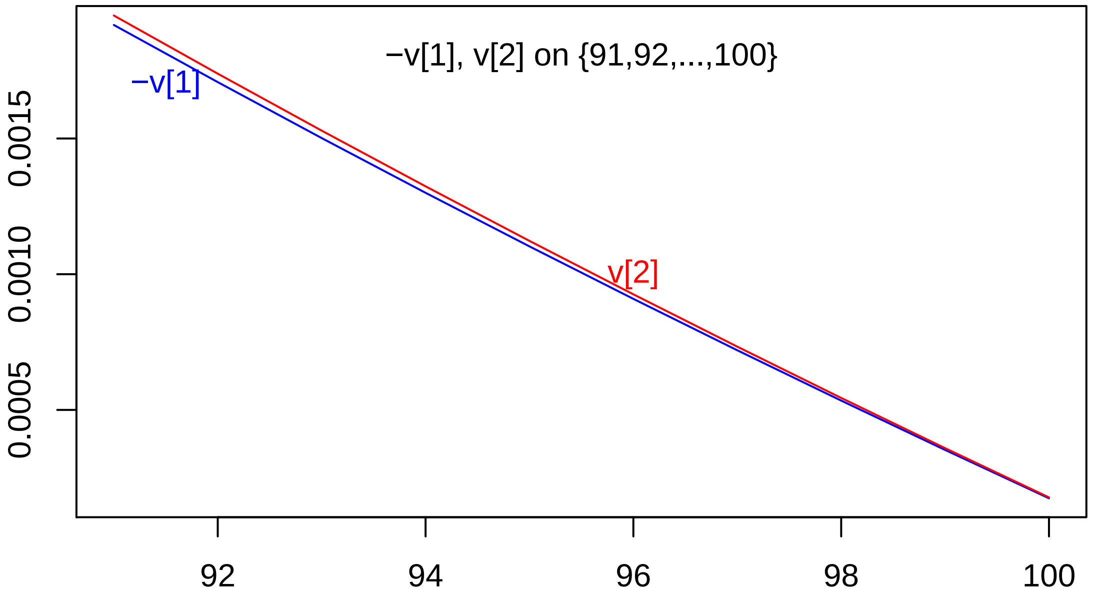
The output here are the approximations $z$ of the maximal eigenvalue of matrix $Q$. However, in what follow, we often write -$z$ instead of $z$ for simplicity.

```
# The approximating sequence z_0, z_1, z_2 of the maximal eigenvalue.
eigenpair$z
```

```
## [1] -0.387333 -0.376393 -0.376383
```

The following figure shows the corresponding eigenvetors $v$ up to a sign. To be consistent with the paper's notation, we accept the convention that the first element in vector start with $v[0]$. From the first figure below, one sees that the eigenvectors $-v[1]$ and $v[2]$ are nearly the same. However, if we look at the last parts of the curves, it is clear that $-v[1]$ and $v[2]$ are actually different.





In the following, because of the space limitation, we do not report the eigenvector results and write -$z$ instead of $z$ for simplicity.

```
# Define the dimension of each matrix.
nn = c(100, 500, 1000, 5000, 7500, 10^4)
```

```
zmat = matrix(0, length(nn), 4)
zmat[, 1] = nn

for (i in 1:length(nn)) {
    # Generate the corresponding tridiagonal matrix for different dimensions.
    a = c(1:(nn[i] - 1))^2
    b = c(1:(nn[i] - 1))^2
    c = rep(0, length(a) + 1)
    c[length(a) + 1] = nn[i]^2

    # Output the corresponding dual case results, i.e, the minimal eigenvalue of
    # the tridiagonal matrix -Q with non-posivie off-diagonal elements.
    zmat[i, -1] = -eff.ini.maxeig.tri(a, b, c, xi = 7/8, improved = T)$z[1:3]
}

colnames(zmat) = c("Dimension", "-z_0", "-z_1", "-z_2")
zmat
# The approximating sequence -z_0, -z_1, -z_2 of the maximal eigenvalue.
```

```
##      Dimension     -z_0     -z_1     -z_2
## [1,]       100 0.387333 0.376393 0.376383
## [2,]       500 0.349147 0.338342 0.338329
## [3,]      1000 0.338027 0.327254 0.327240
## [4,]      5000 0.319895 0.308550 0.308529
## [5,]      7500 0.316529 0.304942 0.304918
## [6,]     10000 0.314370 0.302586 0.302561
```

### The general case (Example 20 in Chen (2016))

The matrix we considered for this general case is

```
# Generate the general matrix A
A = matrix(c(1, 3, 9, 5, 2, 14, 10, 6, 0, 11, 11, 7, 0, 0, 1, 8), 4, 4)
A
```

```
##      [,1] [,2] [,3] [,4]
## [1,]    1    2    0    0
## [2,]    3   14   11    0
## [3,]    9   10   11    1
## [4,]    5    6    7    8
```

This matrix has complex eigenvalues:

```
# Calculate the corresponding eigenvalues of matrix A
eigen(A)$values
```

```
## [1] 24.029261+0.000000i  7.722536+0.000000i  1.124102+2.405217i
## [4]  1.124102-2.405217i
```

We have to be carefull to choice the initial eigenpairs $z_0$ and $v[0]$ for this matrix A. The following is one counterexample with dangerous initials. The approximating sequence converges to the next to maximal eigenvalue rather than the maximal eigenvalue.

```
# Calulate the approximating sequence of maximal eigenvalue by the Rayleigh
# quotient iteration.
eff.ini.maxeig.general(A, z0 = "Auto", digit.thresh = 4)$z
```

```
## [1] "Adjust the diagonal of matrix by Q=A-mI"
## [1] "m=31"
## [1] 13.7532  7.1098  7.6088  7.7219  7.7225
```

Here we fixed the problem by using improved algorithm and safer initials which will be illustrated in detail of the algorithm part.

```
# Calulate the approximating sequence of maximal eigenvalue by the Rayleigh
# quotient iteration.
eff.ini.maxeig.general(A, improved = T, xi = 0.65, digit.thresh = 4)$z
```

```
## [1] "Adjust the diagonal of matrix by Q=A-mI"
## [1] "m=31"
## [1] 24.3016 23.8218 24.0319 24.0293
```

Having illustrated the examples, we now introduce the package in details. It offers four main algorithms:

- `eff.ini.maxeig.tri()`: Calculate the maximal eigenpair for the tridiagonal matrix. The coefficient `xi` is used in the improved initial $z_0$ to form the convex combination of $1/\delta_1$ and $(v_0, Av_0)_\mu$ (in the dual case $(v_0, -Av_0)_\mu$), it should between 0 and 1. The inner product $(,)_\mu$ here is in the space $L^2(\mu)$. The option `improved = F` indicates whether the improved algorithm to find the efficient initials are considered. Usually the option `improved = T` is used when the coefficient `xi` does not equal to 1. The choice `xi=1` is very safe but less effective. When `xi` is a little away from 1, the algorithm becomes more effective. However, when `xi` is closed to 0, the algorithm becomes less effective again and is even dangerous. The "best" choice of `xi` depends on the model. For different models, one may choose different `xi`. In the present tridiagonal case, we suggest a common choice `xi=7/8` (See Example 7 above/below).
  - The precise level used for output results is set to be `digit.thresh = 6` which implies 1e-6 without any special requirement. Same for the following three algorithms and the examples shown in this vignette.
  - This algorithm works not only for tridiagonal matrix, but also for diagonally dominant matrices. In the latter case, one simply picks up the diagonal part from the original one, as illustrated by the last part of Example 17 in the paper Chen (2016). The improved algorithm below Example 10 in the cited paper is recommended.

- `eff.ini.maxeig.general()`: Calculate the maximal eigenpair for the general matrix with non-negative off-diagonal elements. Let the general matrix $A = (a_{ij} : i, j \in E)$.
  - The initial vector $v_0$ is given by the option `v0_tilde`, if `v0_tilde=NULL`, the one computed in Section 4.2 in the paper Chen (2016) is used. Typically, we use two different initial vectors $v_0$, the uniformly distributed one (Section 4.1) and the one computed in Section 4.2, below (10). The first one is safer but less effective and we encourage to use the second one.
  - There are three choices of the initial $z_0$ as the approximation of $\rho(A)$ which is the maximal eigenvalue of the general matrix $A$. The option `z_0="fixed"` corresponding to the choice I, i.e, set $z_0 = \sup_{i\in E} A_i$, where $A_i = \sum_{j\in E} a_{ij}$. The option `z_0="Auto"` corresponding to the choice II, i.e, set $z_0 = v_0^* A v_0$. This simpler choice is used in the other $(z_k : k \geq 1)$, but it may be dangerous as initial $z_0$ as illustrated by Example 20 above/below. The option `z_0="numeric"` corresponding to the choice III, as mentioned in the first algorithm for diagonally dominant matrices.

  - The option `z0numeric` is only used when the initial $z_0$ is considered based on (11) in the paper Chen (2016).
  - Similarly, there are other options `improved`, `xi` and `digit.thresh` which are the same as defined in function `eff.ini.maxeig.tri()`. The improved algorithm is recommended. However, here we should choose a smaller `xi`, say 1/3 or 0.65 used in Examples 18-20 below.

- `eff.ini.secondeig.tri()`: Calculate the next to maximal eigenpair for the tridiagonal matrix. As mentioned in the cited paper, for simplicity, here we assume that the sums of each row of the input tridiagonal matrix should be 0, i.e, $A_i = 0$ for all $i \in E$. Similarly, there are other options `xi` and `digit.thresh` which are the same as defined in function `eff.ini.maxeig.tri()`.

- `eff.ini.secondeig.general()`: Calculate the next to maximal eigenpair for the general conservative matrix where the conservativity of matrix means that the sums of each row are all 0, i.e, $A_i = 0$ for all $i \in E$.

  - There are two choices of the initial $z_0$ as the approximation of $\lambda_1(A)$ which is the second largest eigenvalue of the general matrix $A$. The option `z_0="fixed"` corresponding to the approximation to be $z_0 \approx \lambda_0(A_1)$, where $A_1$ is an auxiliary matrix $A$-matrix given in below. The option `z_0="Auto"` corresponding to the approximation given in Section 6 in the paper Chen (2016). It may be necessary to use `z_0="fixed"`, especially for large matrices.
  - A large constant $c_1$ is provided to replace the last diagonal element $A[N, N]$ to generate the auxiliary matrix $A$-matrix $A_1$. Similarly, there is an option `digit.thresh` which is the same as defined in function `eff.ini.maxeig.tri()`.

There are two auxiliary functions `tridia()` and `ray.quot()` where:

- `tridiag()`: Generate tridiagonal matrix $A$ based on three input vectors.
- `ray.quot()`: Rayleigh quotient iteration algorithm for computing the maximal eigenpair of matrix $A$.

## Acknowledgement

The research project is supported in part by the National Natural Science Foundation of China (No. 11131003), the "985" project from the Ministry of Education in China, and the Project Funded by the Priority Academic Program Development of Jiangsu Higher Education Institutions. The first author acknowledges the assisted one for his hard work in preparing this package in R.

## Examples

We show most of the examples in the paper Chen (2016) in this section. For a convenient comparison, we keep the same subsection names and examples as the paper Chen (2016).

### Efficient initials. The tridiagonal case.

In this subsection, armed with `eff.ini.maxeig.tri()`, we can calculate the maximal eigenpair for the tridiagonal matrix. Sometimes we will report the minimal eigenvalue of $-Q$ by the convention in the paper Chen (2016).

### The minimal eigenvalue of -Q Example 7 (Same as in Example 1)

```
a = c(1:7)^2
b = c(1:7)^2
c = rep(0, length(a) + 1)
c[length(a) + 1] = 8^2
-eff.ini.maxeig.tri(a, b, c, xi = 1)$z
```

```
## [1] 0.485985 0.525313 0.525268
```

### Example 7 (Ordinary and Improved)

In the tridiagonal case, the improved algorithm is presented below Example 10 in Chen (2016).

```
nn = c(100, 500, 1000, 5000, 7500, 10^4)
for (i in 1:6) {
    a = c(1:(nn[i] - 1))^2
    b = c(1:(nn[i] - 1))^2
    c = rep(0, length(a) + 1)
    c[length(a) + 1] = nn[i]^2

    print(-eff.ini.maxeig.tri(a, b, c, xi = 1)$z)
    print(-eff.ini.maxeig.tri(a, b, c, xi = 7/8, improved = T)$z)
}
```

```
## [1] 0.348549 0.376437 0.376383
## [1] 0.387333 0.376393 0.376383
## [1] 0.310195 0.338402 0.338329
## [1] 0.349147 0.338342 0.338329
## [1] 0.299089 0.327320 0.327240
## [1] 0.338027 0.327254 0.327240
## [1] 0.281156 0.308623 0.308529
## [1] 0.319895 0.308550 0.308529
## [1] 0.277865 0.305016 0.304918
## [1] 0.316529 0.304942 0.304918
## [1] 0.275762 0.302660 0.302561
## [1] 0.314370 0.302586 0.302561
```

### The maximal eigenvalue of A Example 8 (Due to L.K.Hua)

```
a = 14/100
b = 40/100
c = c(-25/100 - 40/100, -12/100 - 14/100)
eff.ini.maxeig.tri(a, b, c, xi = 1)$z
eff.ini.maxeig.tri(a, b, c, xi = 7/8, improved = T)$z
```

```
## [1] "Input vector c should be all nonnegative!"
## [1] "Adjust the diagonal of matrix by Q=A-mI"
## [1] "m=0.65"
## [1] 0.437923 0.430603 0.430408
## [1] "Input vector c should be all nonnegative!"
## [1] "Adjust the diagonal of matrix by Q=A-mI"
## [1] "m=0.65"
## [1] 0.436733 0.430407 0.430408
```

**Example 10 (tridiagonal case)**

```
a = c(sqrt(10), sqrt(130)/11)
b = c(11/sqrt(10), 20 * sqrt(130)/143)
c = c(-1 - 11/sqrt(10), -25/11 - sqrt(10) - 20 * sqrt(130)/143, -8/11 - sqrt(130)/11)
eff.ini.maxeig.tri(a, b, c, xi = 1, digit.thresh = 5)$z
eff.ini.maxeig.tri(a, b, c, xi = 7/8, improved = T, digit.thresh = 5)$z
```

```
## [1] "Input vector c should be all nonnegative!"
## [1] "Adjust the diagonal of matrix by Q=A-mI"
## [1] "m=7.02965587709025"
## [1] 5.43937 5.23997 5.23608 5.23607
## [1] "Input vector c should be all nonnegative!"
## [1] "Adjust the diagonal of matrix by Q=A-mI"
## [1] "m=7.02965587709025"
## [1] 5.36161 5.23579 5.23607
```

**Efficient initials. The general case.**

To get the maximal eigenpair for the general matrix $Q$, there are several choices of the initial vector $v_0$ and the initial $z_0$. In this subsection, we study several combinations of the choices.

**Fixed Uniformly distributed initial vector $v_0 = (1, \ldots, 1)/\sqrt(N+1)$.**

**Choice I for $z_0 = \sup_{i \in E} A_i$**

Example 13

```
A = matrix(c(1, 1, 3, 2, 2, 2, 3, 1, 1), 3, 3)
eff.ini.maxeig.general(A, v0_tilde = rep(1, dim(A)[1]), z0 = "fixed", digit.thresh = 5)$z
```

```
## [1] "Adjust the diagonal of matrix by Q=A-mI"
## [1] "m=6"
## [1] 6.00000 5.27273 5.23639 5.23607
```

Example 14

```
A = t(matrix(seq(1, 16), 4, 4))
eff.ini.maxeig.general(A, v0_tilde = rep(1, dim(A)[1]), z0 = "fixed", digit.thresh = 4)$z
```

```
## [1] "Adjust the diagonal of matrix by Q=A-mI"
## [1] "m=58"
## [1] 58.0000 37.3442 36.2674 36.2095 36.2094
```

Example 15

```
A = matrix(c(1, 3, 9, 5, 2, 14, 10, 6, 0, 11, 11, 7, 0, 0, 1, 8), 4, 4)
eff.ini.maxeig.general(A, v0_tilde = rep(1, dim(A)[1]), z0 = "fixed", digit.thresh = 4)$z
```

```
## [1] "Adjust the diagonal of matrix by Q=A-mI"
## [1] "m=31"
## [1] 31.0000 24.4393 24.0385 24.0293
```

Example 16 (Same as Example 1)

```
a = c(1:7)^2
b = c(1:7)^2
c = rep(0, length(a) + 1)
c[length(a) + 1] = 8^2

N = length(a)
Q = tridiag(b, a, -c(b[1] + c[1], a[1:N - 1] + b[2:N] + c[2:N], a[N] + c[N +
    1]))

A = 113 * diag(1, (N + 1)) + Q

113 - eff.ini.maxeig.general(A, v0_tilde = rep(1, dim(A)[1]), z0 = "fixed")$z
```

```
## [1] "Adjust the diagonal of matrix by Q=A-mI"
## [1] "m=113"
## [1] 0.000000 0.602312 0.525463 0.525268
```

**Choice II for $z_0 = v_0^* A v_0$ for Example 13-15**

Example 13

```
A = matrix(c(1, 1, 3, 2, 2, 2, 3, 1, 1), 3, 3)
eff.ini.maxeig.general(A, v0_tilde = rep(1, dim(A)[1]), z0 = "Auto", digit.thresh = 5)$z
```

```
## [1] "Adjust the diagonal of matrix by Q=A-mI"
## [1] "m=6"
## [1] 5.33333 5.24183 5.23608 5.23607
```

Example 14

```
A = t(matrix(seq(1, 16), 4, 4))
eff.ini.maxeig.general(A, v0_tilde = rep(1, dim(A)[1]), z0 = "Auto", digit.thresh = 4)$z
```

```
## [1] "Adjust the diagonal of matrix by Q=A-mI"
## [1] "m=58"
## [1] 34.0000 35.8428 36.2127 36.2094
```

Example 15

```
A = matrix(c(1, 3, 9, 5, 2, 14, 10, 6, 0, 11, 11, 7, 0, 0, 1, 8), 4, 4)
eff.ini.maxeig.general(A, v0_tilde = rep(1, dim(A)[1]), z0 = "Auto", digit.thresh = 4)$z
```

```
## [1] "Adjust the diagonal of matrix by Q=A-mI"
## [1] "m=31"
## [1] 22.0000 23.7316 24.0317 24.0293
```

### Choice III for numeric $z_0$

Here we use the combination coefficient `xi=7/8` which is a little different with Chen (2016).

Example 17: Symmetrizing matrix $A$

```
A = matrix(c(1, 3, 9, 5, 2, 14, 10, 6, 0, 11, 11, 7, 0, 0, 1, 8), 4, 4)
S = (t(A) + A)/2
N = dim(S)[1]
a = diag(S[-1, -N])
b = diag(S[-N, -1])
c = rep(NA, N)
c[1] = -diag(S)[1] - b[1]
c[2:(N - 1)] = -diag(S)[2:(N - 1)] - b[2:(N - 1)] - a[1:(N - 2)]
c[N] = -diag(S)[N] - a[N - 1]

z0ini = eff.ini.maxeig.tri(a, b, c, xi = 7/8, improved = T)$z[1]
z0ini
eff.ini.maxeig.general(A, v0_tilde = rep(1, dim(A)[1]), z0 = "numeric", z0numeric = 28 -
    z0ini, digit.thresh = 4)$z
```

```
## [1] "Input vector c should be all nonnegative!"
## [1] "Adjust the diagonal of matrix by Q=A-mI"
## [1] "m=27"
## [1] 23.70914
## [1] "Adjust the diagonal of matrix by Q=A-mI"
## [1] "m=31"
## [1] 26.7091 24.2703 24.0316 24.0293
```

Example 17: Without Symmetrizing matrix $A$

```
A = matrix(c(1, 3, 9, 5, 2, 14, 10, 6, 0, 11, 11, 7, 0, 0, 1, 8), 4, 4)
S = A
N = dim(S)[1]
a = diag(S[-1, -N])
b = diag(S[-N, -1])
c[1] = -diag(S)[1] - b[1]
c[2:(N - 1)] = -diag(S)[2:(N - 1)] - b[2:(N - 1)] - a[1:(N - 2)]
c[N] = -diag(S)[N] - a[N - 1]

z0ini = eff.ini.maxeig.tri(a, b, c, xi = 7/8, improved = T)$z[1]
z0ini
eff.ini.maxeig.general(A, v0_tilde = rep(1, dim(A)[1]), z0 = "numeric", z0numeric = 31 -
    z0ini, digit.thresh = 4)$z
```

```
## [1] "Input vector c should be all nonnegative!"
## [1] "Adjust the diagonal of matrix by Q=A-mI"
## [1] "m=28"
## [1] 23.68064
## [1] "Adjust the diagonal of matrix by Q=A-mI"
## [1] "m=31"
## [1] 23.6806 23.9865 24.0293
```

### Efficient initial vector $v_0$

The improved algorithm is presented at the end of Section 4 in Chen (2016). The outputs at the last line in Examples 18 and 19 are different from those presented in the cited paper, due to the reason that a slight different combination was used there.

#### Example 18 (Same as Example 10)

```
A = matrix(c(1, 1, 3, 2, 2, 2, 3, 1, 1), 3, 3)
eff.ini.maxeig.general(A, z0 = "Auto", digit.thresh = 5)$z
eff.ini.maxeig.general(A, improved = T, xi = 1, digit.thresh = 5)$z
eff.ini.maxeig.general(A, improved = T, xi = 1/3, digit.thresh = 5)$z
```

```
## [1] "Adjust the diagonal of matrix by Q=A-mI"
## [1] "m=6"
## [1] 5.11616 5.23883 5.23607
## [1] "Adjust the diagonal of matrix by Q=A-mI"
## [1] "m=6"
## [1] 5.90955 5.22251 5.23611 5.23607
## [1] "Adjust the diagonal of matrix by Q=A-mI"
## [1] "m=6"
## [1] 5.38062 5.23289 5.23607
```

#### Example 19 (Same as Example 14)

```
A = t(matrix(seq(1, 16), 4, 4))
eff.ini.maxeig.general(A, z0 = "Auto", digit.thresh = 4)$z
eff.ini.maxeig.general(A, improved = T, xi = 1, digit.thresh = 4)$z
eff.ini.maxeig.general(A, improved = T, xi = 1/3, digit.thresh = 4)$z
```

```
## [1] "Adjust the diagonal of matrix by Q=A-mI"
## [1] "m=58"
## [1] 34.4924 36.1469 36.2095 36.2094
## [1] "Adjust the diagonal of matrix by Q=A-mI"
## [1] "m=58"
## [1] 57.1959 35.6097 36.2168 36.2094
## [1] "Adjust the diagonal of matrix by Q=A-mI"
## [1] "m=58"
## [1] 41.6238 36.0380 36.2101 36.2094
```

#### Example 20 (Same as Example 15)

```
A = matrix(c(1, 3, 9, 5, 2, 14, 10, 6, 0, 11, 11, 7, 0, 0, 1, 8), 4, 4)
eff.ini.maxeig.general(A, z0 = "Auto", digit.thresh = 4)$z
eff.ini.maxeig.general(A, improved = T, xi = 1, digit.thresh = 4)$z
eff.ini.maxeig.general(A, improved = T, xi = 0.65, digit.thresh = 4)$z
```

```
## [1] "Adjust the diagonal of matrix by Q=A-mI"
## [1] "m=31"
## [1] 13.7532  7.1098  7.6088  7.7219  7.7225
## [1] "Adjust the diagonal of matrix by Q=A-mI"
## [1] "m=31"
## [1] 30.0766 20.0926 24.9028 24.0846 24.0295 24.0293
## [1] "Adjust the diagonal of matrix by Q=A-mI"
## [1] "m=31"
## [1] 24.3016 23.8218 24.0319 24.0293
```

**Example 21**

```
b4 = c(0.01, 1, 100)
digits = c(9, 7, 6)

for (i in 1:3) {
    A = matrix(c(-3, 4, 0, 10, 0, 2, -7, 5, 0, 0, 0, 3, -5, 0, 0, 1, 0, 0, -16,
        11, 0, 0, 0, 6, -11 - b4[i]), 5, 5)
    print(-eff.ini.maxeig.general(A, z0 = "Auto", digit.thresh = digits[i])$z)
}
```

```
## [1] 0.002000000 0.000278151 0.000278686
## [1] 0.0990974 0.0236258 0.0245174 0.0245175
## [1] 1.666906 0.200058 0.182609 0.182819
```

**Example 22**

The result given by general algorithm

```
b4 = c(0.01, 1, 100)
digits = c(9, 7, 6)

for (i in 1:3) {
    A = matrix(c(-5, 3, 0, 0, 0, 5, -7, 2, 0, 0, 0, 4, -3, 10, 0, 0, 0, 1, -16,
        11, 0, 0, 0, 6, -11 - b4[i]), 5, 5)
    print(-eff.ini.maxeig.general(A, z0 = "Auto", digit.thresh = digits[i])$z)
}
```

```
## [1] 0.002000000 0.000278151 0.000278686
## [1] 0.1091040 0.0234222 0.0245174 0.0245175
## [1] 1.550170 0.133420 0.182541 0.182819
```

The result given by tri algorithm

```
b4 = c(0.01, 1, 100)
a = c(3, 2, 10, 11)
b = c(5, 4, 1, 6)
b4 = c(0.01, 1, 100, 10^6)
digits = c(9, 7, 6, 6)

for (i in 1:4) {
    c = c(rep(0, 4), b4[i])
    print(-eff.ini.maxeig.tri(a, b, c, xi = 1, digit.thresh = digits[i])$z)
}
```

```
## [1] 0.000278670 0.000278686
## [1] 0.0244003 0.0245190 0.0245175
## [1] 0.179806 0.182912 0.182819
## [1] 0.191917 0.195239 0.195145
```

**The next to the maximal eigenpair. Tridiagonal matrix case.**

**Example 25**

```
a = c(1:7)^2
b = c(1:7)^2

eff.ini.seceig.tri(a, b, xi = 0)$z
eff.ini.seceig.tri(a, b, xi = 1)$z
eff.ini.seceig.tri(a, b, xi = 2/5)$z

## [1] 0.902633 0.820614 0.820539
## [1] 0.456343 0.821600 0.820539
## [1] 0.724117 0.820629 0.820539
```

**Example 26**

```
a = c(3, 2, 10, 11)
b = c(5, 4, 1, 6)

eff.ini.seceig.tri(a, b, xi = 0, digit.thresh = 5)$z
eff.ini.seceig.tri(a, b, xi = 1, digit.thresh = 5)$z
eff.ini.seceig.tri(a, b, xi = 2/5, digit.thresh = 5)$z

## [1] 3.84977 3.05196 3.03673
## [1] 1.72924 3.05715 3.03673
## [1] 3.00156 3.03675 3.03673
```

**The next to the maximal eigenpair. General matrix case.**

The results of two choices of the initial $z_0$ are presented in the following.

**Example 27**

```
Q = matrix(c(-30, 1/5, 11/28, 55/3291, 30, -17, 275/42, 330/1097, 0, 84/5, -20,
    588/1097, 0, 0, 1097/84, -2809/3291), 4, 4)
eff.ini.seceig.general(Q, z0 = "Auto", digit.thresh = 5)$z
eff.ini.seceig.general(Q, z0 = "fixed", digit.thresh = 5)$z

## [1] 7.73666 8.15020 8.17129 8.17131
## [1] 7.34195 8.13214 8.17124 8.17131
```

**Example 28**

```
Q = matrix(c(-57, 135/59, 243/91, 351/287, 118/27, -52, 590/91, 118/41, 91/9,
    637/59, -47, 195/41, 1148/27, 2296/59, 492/13, -62/7), 4, 4)
eff.ini.seceig.general(Q, z0 = "Auto", digit.thresh = 4)$z
eff.ini.seceig.general(Q, z0 = "fixed", digit.thresh = 4)$z

## [1] 47.5318 47.5453 47.5454
## [1] 38.7143 47.5346 47.5453 47.5454
```