Tokyo Woman's Christian University
東京女子大学

Verified Numerical Computations
VNC-HPC
High-Performance Computing

数値線形代数における前処理と反復改良法

荻田 武史
東京女子大学 現代教養学部 数理科学科

第17回AT研究会オープンアカデミックセッション(ATOS17)
山梨大学
2017/10/6

概要

数値線形代数

- 連立一次方程式, 行列固有値問題, 特異値問題など

HPC環境等における問題の大規模化・複雑化 $\Rightarrow$ 高精度な解を得ることが困難（問題の悪条件化）

$$\Downarrow$$

本発表では、問題が悪条件な場合に有効な前処理技法と反復改良法について議論する。

- 一般密行列系の連立一次方程式
- 対称系の固有値問題

連立一次方程式

Outline

Let us consider solving a linear system

$$Ax = b, \quad A \in \mathbb{R}^{n \times n},\ b \in \mathbb{R}^n \tag{1}$$

using floating-point arithmetic.

$\Longrightarrow$ One of the significant tasks in scientific computing

$\mathbf{u}$: rounding error unit ($\mathbf{u} = 2^{-53} \approx 10^{-16}$ for IEEE 754 binary64)

$\boxed{\text{Purpose}}$ Develop efficient methods for solving (1) in ill-conditioned cases such that

$$\kappa(A) \gtrsim \mathbf{u}^{-1}.$$

$\Rightarrow\ \kappa(A) \gtrsim 10^{16}$ for IEEE 754 binary64 $\Rightarrow$ Hard to solve!

Difficulty

- $\mathbf{u}$: rounding error unit ($\mathbf{u} = 2^{-53} \approx 10^{-16}$ for IEEE 754 binary64)
- $\kappa(A) := \|A\| \cdot \|A^{-1}\|$: condition number
- $x := A^{-1}b$: exact solution of $Ax = b$
- $\widehat{x}$: approximate solution

$\boxed{\text{Rule of thumb}}$ Standard numerical algorithms in floating-point satisfy

$$\frac{\|\widehat{x} - x\|}{\|x\|} \approx p(n) \cdot \kappa(A) \cdot \mathbf{u} \ .$$

$\Longrightarrow$ Inaccurate approximate solutions are produced if $\kappa(A) \gtrsim \mathbf{u}^{-1}$.

Iterative refinement

1. Execute an LU factorization of A.

2. Compute an approximate solution $\widehat{x}$ of $Ax = b$ using LU factors.

3. Iterate the following process:

   (a) Compute the residual $r = b - A\widehat{x}$ in higher-precision arithmetic (usually, twice the working precision is sufficient).
   (b) Compute an approximate solution $\widehat{y}$ of $Ay = r$ using LU factors.
   (c) Update $\widehat{x}$ by $\widehat{x} \leftarrow \widehat{x} + \widehat{y}$.

$$\Longrightarrow \text{This cannot work for } \kappa(A) \gtrsim \mathbf{u}^{-1}.$$

Why?

Why cannot iterative refinement work in ill-conditioned cases?

Suppose $Ay = r$ is solved with LU factors. Then we have

$$\frac{\|\widehat{y} - y\|}{\|y\|} \approx p(n) \cdot \kappa(A) \cdot \mathbf{u} \ .$$

$\Longrightarrow$ If $\kappa(A) \gtrsim \mathbf{u}^{-1}$, then $\widehat{y}$ is likely to be inaccurate.

$\Longrightarrow$ $\widehat{x}$ is not refined.

$\Longrightarrow$ We need to overcome the condition number!

Why not simply using multiple precision arithmetic?

- MP is significantly slower than hardware (binary32/64) arithmetic.
  $\Longrightarrow$ We prefer to utilize binary64 arithmetic in terms of computational speed.

- Usually, standard numerical algorithms provide good results in binary64 arithmetic.
  $\Longrightarrow$ The use of MP may not be necessary.

- We do not know in advance how many digits suffice to achieve desired result accuracy.
  $\Longrightarrow$ We prefer some preconditioning method utilizing the results obtained by a standard numerical algorithm (LU factorization with partial pivoting).

Background

We adopt an approach of preconditioning methods.

$$\kappa(MA) \approx \mathbf{u} \cdot \kappa(A), \quad M \in \mathbb{R}^{n \times n}$$

$\boxed{\text{Approximate inverse of } A \text{ as a choice of } M}$ Rump (1980's, 2009, 2013) showed that for a matrix A with $\kappa(A) > \mathbf{u}^{-1}$ an approximate inverse R of A obtained in pure fl-pt arithmetic still contains some useful information:

- Useless as the inverse of A, $\quad \|I - RA\| \not< 1$
- Useful as a preconditioner for A, $\quad \kappa(RA) \approx \mathbf{u} \cdot \kappa(A)$

$\boxed{\text{Modification}}$

Approximate inverse of an LU factor of A (O., 2010, Kobayashi–O., 2015+) $\Longrightarrow$ less computational cost (about $1/2$)

Previous work: Approximate inverse of an LU factor

Suppose $A \approx LU$ (Crout's version) with $\kappa(A) \approx \kappa(L)$, and put $M := L^{-1}$.

- $\kappa(L^{-1}A) \approx \mathbf{u} \cdot \kappa(A)$. [O., SIMAX 2010]
- Higher-precision arithmetic is necessary for $X_L \cdot A$ with $X_L \approx L^{-1}$.

Major computational cost in $\mathcal{O}(n^3)$ flops:

- LU factorization $A \approx LU$. $\Rightarrow$ $\frac{2}{3}n^3$ flops
- Compute $X_L \approx L^{-1}$. $\Rightarrow$ $\frac{1}{3}n^3$ flops
- Compute $X_L \cdot A$ in twice the working precision.
  $\Rightarrow$ $c \cdot n^3$ flops with relatively large constant $c = 10 \sim 22$

$\boxed{\text{We aim to reduce the cost.}}$

Idea

$$\boxed{\text{Do we need full information of } L^{-1}?}$$

$\Longrightarrow$ It depends on the distribution of the singular values of A.

An example: only one large singular value

```
>> n=3; cnd=1e15; A=gallery('randsvd',n,cnd,1); svd(A)
ans =
   1.0000e+00
   1.0402e-15
   1.0003e-15

>> double(cond(mp(A)))
ans =
   9.9905e+14
```

```
>> [UT,LT,P]=lu(A'); L=LT' % P*A' = UT*LT
L =
  -7.0193e-01            0            0
   5.4324e-01  -1.2351e-15            0
  -2.6479e-01   5.5511e-16   1.1752e-15

>> XL=eye(n)/L % Solve XL*L = I for XL
XL =
  -1.4247e+00            0            0
  -6.2660e+14  -8.0964e+14            0
  -2.5014e+13   3.8242e+14   8.5089e+14

>> double(cond(mp(XL)*mp(A)))
ans =
   1.6569e+00
```

```
>> L2=L; Eps=1e-16; L2(2:3,2:3)=Eps*tril(randn(2))
L2 =
  -7.0193e-01             0             0
   5.4324e-01   7.2540e-17             0
  -2.6479e-01  -6.3055e-18  -2.0497e-17

>> XL2=eye(n)/L2
XL2 =
  -1.4247e+00             0             0
   1.0669e+16   1.3785e+16             0
   1.5122e+16  -4.2409e+15  -4.8789e+16

>> double(cond(mp(XL2)*mp(A)))
ans =
   5.5197e+01
```

```
>> L3=L; Eps=1e-16; L3(2:3,2:3)=Eps*eye(2)
L3 =
  -7.0193e-01             0             0
   5.4324e-01   1.0000e-16             0
  -2.6479e-01             0   1.0000e-16

>> XL3=eye(n)/L3
XL3 =
  -1.4247e+00             0             0
   7.7393e+15   1.0000e+16             0
  -3.7723e+15             0   1.0000e+16

>> double(cond(mp(XL3)*mp(A)))
ans =
   1.3710e+01
```

The proposed method

α: threshold $(\mathbf{u} \le \alpha \le 1)$

$\boxed{\text{Special case}}$ one large l_{ii}, $\quad |l_{ii}| < \alpha\|A\|,\ i \ge 2$

$$L = \begin{bmatrix} l_{11} & O \\ w & * \end{bmatrix} \to \widetilde{L} := \begin{bmatrix} l_{11} & O \\ w & \alpha I_{n-1} \end{bmatrix} \Rightarrow \widetilde{L}^{-1} = \begin{bmatrix} 1/l_{11} & O \\ -\frac{1}{\alpha} w/l_{11} & \frac{1}{\alpha} I_{n-1} \end{bmatrix}$$

$\boxed{\text{Generalization}}$ m large l_{ii}, $\begin{cases} |l_{ii}| \ge \alpha\|A\| & i \le m \\ |l_{ii}| < \alpha\|A\| & i \ge m+1 \end{cases}$

$$L = \begin{bmatrix} L_{11} & O \\ L_{21} & * \end{bmatrix} \to \widetilde{L} := \begin{bmatrix} L_{11} & O \\ L_{21} & \alpha I_{n-m} \end{bmatrix} \Rightarrow \widetilde{L}^{-1} = \begin{bmatrix} L_{11}^{-1} & O \\ -\frac{1}{\alpha} L_{21} L_{11}^{-1} & \frac{1}{\alpha} I \end{bmatrix}$$

$\boxed{\text{Observation}}$ $\underline{\underline{\kappa(\widetilde{L}^{-1}A) \approx \alpha \cdot \kappa(A)}}$

Remarks

- We may stop the LU factorization of A at the m-th step if $|l_{m+1,m+1}| < \alpha\|A\|$.
  $\Rightarrow$ Further computations become meaningless.

- It seems not possible to develop an analogous method using approximate inverse R of A, since all the elements of R are equilibrated in magnitude.

```
>> R=inv(A)
R =
   9.8881e+13  -4.5588e+13  -3.5565e+14
  -2.4722e+13   3.7796e+14   8.4096e+14
  -6.3383e+14  -6.9922e+14   2.4567e+14
```

Application to linear systems

For $A = \begin{bmatrix} A_1 \\ A_2 \end{bmatrix}$, $U = \begin{bmatrix} U_1 \\ U_2 \end{bmatrix}$, $b = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$,

$$\begin{aligned}
\widetilde{L}^{-1}A &= \begin{bmatrix} I & O \\ O & \frac{1}{\alpha}I \end{bmatrix} \begin{bmatrix} L_{11}^{-1} & O \\ -L_{21}L_{11}^{-1} & I \end{bmatrix} \begin{bmatrix} A_1 \\ A_2 \end{bmatrix} \\
&= \begin{bmatrix} I & O \\ O & \frac{1}{\alpha}I \end{bmatrix} \begin{bmatrix} L_{11}^{-1}A_1 \\ A_2 - L_{21}L_{11}^{-1}A_1 \end{bmatrix} \\
&\approx \begin{bmatrix} U_1 \\ \frac{1}{\alpha}(A_2 - WA_1) \end{bmatrix} =: A', \quad W \approx L_{21}L_{11}^{-1}
\end{aligned}$$

$$\widetilde{L}^{-1}b = \begin{bmatrix} d_1 \\ \frac{1}{\alpha}(b_2 - Wb_1) \end{bmatrix} =: b', \quad d_1 \approx L_{11}^{-1}b_1$$

Higher-precision arithmetic: Dot2 (dot product in 'doubled' working precision)

$$
\begin{array}{ll}
\textbf{function} \;\; c = \texttt{Dot2}(x, y) & \texttt{\%}\ x, y \in \mathbb{F}^n,\ c \in \mathbb{F} \\
[p, s] = \texttt{TwoProduct}(x_1, y_1) & \texttt{\%}\ p + s \leftarrow x_1 \cdot y_1 \\
for\ i = 2 : n & \\
\quad [h, r] = \texttt{TwoProduct}(x_i, y_i) & \texttt{\%}\ h + r \leftarrow x_i \cdot y_i \\
\quad [p, q] = \texttt{TwoSum}(p, h) & \texttt{\%}\ p + q \leftarrow p + h \\
\quad s = s + (r + q) & \\
end & \\
c = p + s &
\end{array}
$$

O., S. M. Rump, S. Oishi: Accurate sum and dot product, SIAM J. Sci. Comput., 26:6 (2005), 1955-1988.

Computational complexity: previous method

Table 1: Previous method

$\mathcal{O}(n^3)$ operations	flops
LU of A	$\frac{2}{3}n^3$
$X_L \approx L^{-1}$	$\frac{1}{3}n^3$
$A' \approx X_L A$ with `Dot2`	$\frac{25}{2}n^3$
LU of A'	$\frac{2}{3}n^3$
Total	$\frac{85}{6}n^3$

Computational complexity: proposed method

Table 2: Proposed method

$\mathcal{O}(n^3)$ operations	flops
LU of A	$\frac{2}{3}n^3$
$W \approx L_{21}L_{11}^{-1}$	$(n-m)m^2$
$A_2 - WA_1$ with `Dot2`	$\frac{25}{2}(n-m)mn$
LU of A'	$\frac{2}{3}n^3$
Total	$\frac{4}{3}n^3 + (\frac{25}{2}n + m)(n-m)m$

approx. maximized at $m = \frac{1}{2}n \;\sim\; \frac{55}{12}n^3 \approx 4.6n^3$ flops

Previous: $\frac{85}{6}n^3 \approx 14.2n^3$ flops

Much less cost even in the worst case!

Further improvement: LU of A and A'

- LU of A can be stopped at the m-th step.

$$\text{flops: } \tfrac{2}{3}n^3 \quad \Rightarrow \quad \tfrac{2}{3}m^3 + 2(n-m)mn$$

- Computational cost for LU of A' can be reduced.

$$\begin{aligned} A' &= \begin{bmatrix} U_1 \\ \frac{1}{\alpha}(A_2 - WA_1) \end{bmatrix} = \begin{bmatrix} U_{11} & U_{12} \\ C_{21} & C_{22} \end{bmatrix} \\ &= \begin{bmatrix} I_m & O \\ C_{21}U_{11}^{-1} & L' \end{bmatrix} \begin{bmatrix} U_{11} & U_{12} \\ O & U' \end{bmatrix} \\ &= \begin{bmatrix} U_{11} & U_{12} \\ C_{21} & C_{21}U_{11}^{-1}U_{12} + L'U' \end{bmatrix} \end{aligned}$$

LU of Schur complement: $C_{22} - C_{21}U_{11}^{-1}U_{12} = L'U'$

Cost: $\boxed{\tfrac{2}{3}(n-m)^3 + m^2(n-m) + 2m(n-m)^2 \text{ flops}}$

Numerical results

Several kinds of singular value distributions

MATLAB's built-in function `randsvd` from Higham's test matrices:

```
>> A = gallery('randsvd',n,cnd,mode,n,n,1);
```

For input arguments `mode` $\in \{1, 2, 3, 4, 5\}$ and `cnd` $=: \beta \geq 1$,

1. one large singular value: $\sigma_1 \approx 1$, $\sigma_i \approx \beta^{-1}$, $i = 2 : n$
2. one small singular value: $\sigma_n \approx \beta^{-1}$, $\sigma_i \approx 1$, $i = 1 : n-1$
3. geometrically distributed: $\sigma_i \approx \beta^{-\frac{i-1}{n-1}}$, $i = 1 : n$
4. arithmetically distributed: $\sigma_i \approx 1 - (1 - \beta^{-1})\dfrac{i-1}{n-1}$, $i = 1 : n$
5. random singular values with uniformly distributed logarithm: $\sigma_i \approx \beta^{-r(i)}$, $i = 1 : n$, where $r(i)$ are pseudo-random values drawn from the standard uniform distribution on $(0, 1)$.

Algorithms

- Previous method ($X_L \cdot A$ using `Dot2`)
- Proposed method

Maximum relative error For $x^* = A^{-1}b$ (practically obtained by multiple precision arithmetic),

$$\max_i \left| \frac{x_i^* - \widehat{x}_i}{x_i^*} \right|$$

Computational environment

- CPU: Intel Xeon E5-4617 2.9GHz (6 cores $\times$ 4)
- Memory: 1TB
- MATLAB R2015b
- Intel C++ Compiler version 13.1.1、gcc version 4.4.7

Proposed method

- Threshold: $\alpha = 10^{-9}$ ($|l_{ii}| < \alpha \|A\|$)

- Iterative refinement
  - Stopping criterion: $|\widehat{x}_i^{(k+1)} - \widehat{x}_i^{(k)}| \leq \epsilon |\widehat{x}_i^{(k+1)}|$ for $\epsilon = 10^{-9}$
  - Max. number of iterations: $k_{\max} = 16$

- `Dot2` is naively implemented using MATLAB MEX with C and OpenMP.

Results ($n = 5000$, $\texttt{mode} = 3$)

Table 3: Computing time (sec.) and its ratio to LU

		Previous method				Proposed method				
cnd	LU	Time	k_1	k_2	r_{LU}	Time	k_1	k_2	r_{LU}	m
10^4	0.96	1.02	1	–	1.1	1.02	1	–	1.1	–
10^8	0.93	1.02	2	–	1.1	1.02	2	–	1.1	–
10^{12}	0.94	1.06	3	–	1.1	1.06	3	–	1.1	–
10^{14}	0.93	1.09	5	–	1.2	1.09	5	–	1.2	–
10^{15}	0.93	19.34	1	1	20.6	6.49	1	3	6.9	3861
10^{16}	0.95	19.33	1	1	20.3	6.20	1	4	6.8	3597

- r_{LU}: Ratio to time of LU (MATLAB's lu)
- Theoretical r_{LU}: $\mathrm{Previous} \approx 21$, $\mathrm{Proposed} \approx 6.2$ ($m = 3500$)

Results ($n = 5000$, $\texttt{mode} = 3$)

Table 4: Maximum relative error

cnd	MATLAB $A\backslash b$	Previous method	Proposed method
10^4	$1.9 \cdot 10^{-11}$	$2.2 \cdot 10^{-16}$	$2.2 \cdot 10^{-16}$
10^8	$4.5 \cdot 10^{-08}$	$2.2 \cdot 10^{-16}$	$2.2 \cdot 10^{-16}$
10^{12}	$2.3 \cdot 10^{-04}$	$2.2 \cdot 10^{-16}$	$2.2 \cdot 10^{-16}$
10^{14}	$1.8 \cdot 10^{-02}$	$2.9 \cdot 10^{-14}$	$2.9 \cdot 10^{-14}$
10^{15}	$2.1 \cdot 10^{-01}$	$4.4 \cdot 10^{-16}$	$1.3 \cdot 10^{-13}$
10^{16}	$2.7 \cdot 10^{+01}$	$5.1 \cdot 10^{-14}$	$6.2 \cdot 10^{-14}$

- tolerance: $\epsilon = 10^{-9}$
- $2^{-52} \approx 2.22 \cdot 10^{-16}$

Results ($n = 5000$, $\texttt{mode} \in \{1, 2, 4, 5\}$)

Table 5: Computing time (sec.) and its ratio to LU

	Proposed method ($\texttt{mode} = 1$)						Proposed method ($\texttt{mode} = 2$)					
cnd	LU	Time	k_1	k_2	r_{LU}	m	LU	Time	k_1	k_2	r_{LU}	m
10^{12}	0.95	1.06	2	–	1.1	–	0.94	1.09	3	–	1.2	–
10^{14}	0.94	1.13	4	–	1.2	–	0.95	3.88	1	1	4.1	4999
10^{15}	0.93	1.14	5	–	1.2	–	0.91	3.78	1	1	4.1	4999
10^{16}	0.94	2.91	1	1	3.1	1	0.92	3.94	2	2	4.3	4999

	Proposed method ($\texttt{mode} = 4$)						Proposed method ($\texttt{mode} = 5$)					
cnd	LU	Time	k_1	k_2	r_{LU}	m	LU	Time	k_1	k_2	r_{LU}	m
10^{12}	0.94	1.08	3	–	1.2	–	0.94	1.09	3	–	1.2	–
10^{14}	0.93	4.24	1	5	4.6	4999	0.94	1.13	4	–	1.2	–
10^{15}	0.92	3.82	1	1	4.2	4999	0.94	6.42	1	3	6.8	3864
10^{16}	0.92	3.79	1	1	4.1	4999	0.94	6.62	1	4	7.1	3589

mode	Theoretical r_{LU}	m
1	2	1
2	2	4999

mode	Theoretical r_{LU}	m
4	2	4999
5	6.2	3500

Latest results with further improvement for LU (experiments done by Y. Kobayashi)

- MATLAB R2016b, 2 CPUs: 3.0 GHz Intel Xeon E5-2687W v4
- To reduce MATLAB's overhead, several parts of computations are converted to MEX in C with BLAS/LAPACK/OpenMP.

Table 6: Results for randsvd, $n = 5000$, $\kappa(A) \approx 10^{16}$, $tol = 10^{-9}$

mode	r_{LU}	Max. rel. error	k_2	m
1	1.21	$4.45 \cdot 10^{-14}$	1	1
2	1.20	$5.84 \cdot 10^{-11}$	3	4999
3	22.06	$6.57 \cdot 10^{-14}$	2	2486
4	1.10	$4.37 \cdot 10^{-12}$	1	4999
5	22.13	$3.49 \cdot 10^{-14}$	2	2436

r_{LU}: Ratio to time for LU factorization (MATLAB's lu)

Future work

- More numerical results
- Detailed analysis

固有値問題

(joint work with Kensuke Aishima)

Outline

Let us consider the standard eigenvalue problem

$$Ax = \lambda x \tag{2}$$

where $A = A^T \in \mathbb{R}^{n \times n}$.

$\Longrightarrow$ One of the significant tasks in scientific computing

$\boxed{\text{Purpose}}$ develop a simple algorithm for computing an arbitrarily accurate eigenvalue decomposition:

$$A = XDX^{-1} = XDX^T,$$

where $X \in \mathbb{R}^{n \times n}$ is orthogonal and $D \in \mathbb{R}^{n \times n}$ is diagonal with $d_{ii} = \lambda_i$.

Difficulty

- μ_i: approximate eigenvalues
- $z_{(i)}$: approximate eigenvectors, $\|z_i\| = 1$
- $gap(\mu_i)$: the smallest difference between μ_i and any other eigenvalue, i.e., $gap(\mu_i) := \min_{j \neq i} |\mu_i - \lambda_j|$.

For eigenvalues:

$$|\lambda_i - \mu_i| \leq \|Az_{(i)} - \mu_i z_{(i)}\|.$$

For eigenvectors:

$$|\sin\theta(x_{(i)}, z_{(i)})| \leq \frac{\|Az_{(i)} - \mu_i z_{(i)}\|}{gap(\mu_i)}, \quad \theta(x_{(i)}, z_{(i)}) := \arccos(|x_{(i)}^{\mathrm{T}} z_{(i)}|).$$

$$\Longrightarrow \ gap \ \rightarrow \ \text{small} \quad \Rightarrow \quad \text{Error of eigenvectors} \ \rightarrow \ \text{large}$$

Note

- We assume A to be dense (not sparse).
- We DO NOT intend to compete with existing eigensolvers such as Householder's reduction + (QR, MRRR, etc.) nor the Jacobi algorithm.
- We develop an iterative refinement algorithm independent of eigensolvers.

$\Longrightarrow$ The proposed algorithm can be regarded as a supplement to the existing eigensolvers.

$\Longrightarrow$ In fact, we assume that some backward stable algorithm is available.

Iterative refinement for eigenvalue problems

Most of the existing refinement methods are based on Newton's method for nonlinear equations:

Consider the function $g : \mathbb{C}^{n+1} \to \mathbb{C}^{n+1}$ for $A \in \mathbb{C}^{n\times n}$ such that

$$g(z) = g\left(\begin{bmatrix} x \\ \lambda \end{bmatrix}\right) = \begin{bmatrix} Ax - \lambda x \\ \|x\| - 1 \end{bmatrix},$$

where $\|\cdot\|$ denotes some vector norm.

Then a solution of $g(z) = \mathbf{0}$ corresponds directly to an eigenpair $(\lambda, x) \in \mathbb{C} \times \mathbb{C}^n$ of the matrix A with $\|x\| = 1$.

Therefore, by applying Newton's method for $g(z) = \mathbf{0}$, we can improve the accuracy of each approximate eigenpair with quadratic convergence.

For example,

[1] J. J. Dongarra, C. B. Moler, J. H. Wilkinson: Improving the accuracy of computed eigenvalues and eigenvectors, SIAM J. Numer. Anal., 20:1 (1983), 23–45.

[2] F. Tisseur: Newton's method in floating point arithmetic and iterative refinement of generalized eigenvalue problems, SIAM J. Matrix Anal. Appl., 22:4 (2001), 1038–1057.

Using these methods we can improve eigenpairs with $\mathcal{O}(n^3)$ operations for each eigenpair.

$\Longrightarrow$ For n eigenpairs, $\mathcal{O}(n^4)$ operations.

$\Longrightarrow$ We try to develop an efficient method of improving all eigenpairs at the same time with $\mathcal{O}(n^3)$ operations.

Basic idea

$X \in \mathbb{R}^{n\times n}$: eigenvector matrix comprising all eigenvectors of A
$X^{(0)} \in \mathbb{R}^{n\times n}$: initial approximation of X

$\boxed{\text{Task}}$ Develop a simple iterative refinement algorithm (in the same manner as Newton's method) for updating $X^{(k)}$, $k = 1, 2, \ldots$ such that

- $X^{(k)} := X^{(k-1)} + W^{(k)}$, $\quad W^{(k)}$: some correction term
- $\|X - X^{(k)}\| = \mathcal{O}(\|X - X^{(k-1)}\|^2)$ (quadratic convergence)

More precisely ...

$X \in \mathbb{R}^{n\times n}$: exact, $\quad \widehat{X} \in \mathbb{R}^{n\times n}$: approximation of X

Define $E \in \mathbb{R}^{n\times n}$ such that $X = \widehat{X}(I + E)$.

$\boxed{\text{Assumption}}$ $\|E\| =: \varepsilon < 1$

$\boxed{\text{Task}}$ Compute a good approximation $\widetilde{E}$ of E.

$\Longrightarrow$ We utilize the following two relations:

$$\begin{cases} X^T X = I & \text{(orthogonality)} \\ X^T A X = D & \text{(diagonality)} \end{cases}$$

Orthogonality: $X^T X = I$

$$
\begin{aligned}
& I = X^T X = (I+E)^T \widehat{X}^T \widehat{X} (I+E) \\
\Leftrightarrow \quad & (I+E)^{-T}(I+E)^{-1} = \widehat{X}^T \widehat{X} \\
\Leftrightarrow \quad & (I - E + \Delta_E)^T (I - E + \Delta_E) = \widehat{X}^T \widehat{X}, \quad \|\Delta_E\| = \mathcal{O}(\varepsilon^2) \\
\Leftrightarrow \quad & E + E^T = I - \widehat{X}^T \widehat{X} + \Delta_1, \quad \|\Delta_1\| = \mathcal{O}(\varepsilon^2)
\end{aligned}
$$

Ignoring Δ_1, we have the equation

$$
\widetilde{E} + \widetilde{E}^T = I - \widehat{X}^T \widehat{X}. \tag{3}
$$

Diagonality: $X^T A X = D$

$$
\begin{aligned}
& D = X^T A X \\
\Leftrightarrow\ & D = (I+E)^T \widehat{X}^T A \widehat{X} (I+E) \\
\Leftrightarrow\ & (I+E)^{-T} D (I+E)^{-1} = \widehat{X}^T A \widehat{X} \\
\Leftrightarrow\ & (I - E + \Delta_E)^T D (I - E + \Delta_E) = \widehat{X}^T A \widehat{X}, \quad \|\Delta_E\| = \mathcal{O}(\varepsilon^2) \\
\Leftrightarrow\ & D - DE - E^T D = \widehat{X}^T A \widehat{X} + \Delta_2, \quad \|\Delta_2\| = \mathcal{O}(\|A\|\varepsilon^2)
\end{aligned}
$$

Ignoring Δ_2, we have the equation

$$\widetilde{D} - \widetilde{D}\widetilde{E} - \widetilde{E}^T \widetilde{D} = \widehat{X}^T A \widehat{X}. \tag{4}$$

Solve the system of matrix equations

Let $R = (r_{ij})$, $S = (s_{ij})$ be defined by

$$R := I - \widehat{X}^T \widehat{X} \quad \text{and} \quad S := \widehat{X}^T A \widehat{X}.$$

We have

$$\begin{cases} \widetilde{E} + \widetilde{E}^T = R & (*) \\ \widetilde{D} - \widetilde{D}\widetilde{E} - \widetilde{E}^T \widetilde{D} = S & (**) \end{cases}.$$

Unknowns: $\widetilde{E} = (\widetilde{e}_{ij})$, $\widetilde{D} = \mathrm{diag}(\widetilde{\lambda}_i)$

After obtaining $E^{(k)} := \widetilde{E}$, we can update $X^{(k)}$ by

$$X^{(k+1)} := X^{(k)}(I + E^{(k)}) = X^{(k)} + X^{(k)} E^{(k)}.$$

First we focus on the diagonal parts:
From (*),

$$\operatorname{diag}(\widetilde{E} + \widetilde{E}^T) = \operatorname{diag}(R) \;\Rightarrow\; \underline{\underline{\widetilde{e}_{ii} = \frac{r_{ii}}{2}}}.$$

From (**),

$$\begin{aligned} & \operatorname{diag}(\widetilde{D} - \widetilde{D}\widetilde{E} - \widetilde{E}^T\widetilde{D}) = \operatorname{diag}(S) \\ \Leftrightarrow\; & \widetilde{\lambda}_i - 2\widetilde{\lambda}_i\widetilde{e}_{ii} = s_{ii} \\ \Leftrightarrow\; & \widetilde{\lambda}_i = \frac{s_{ii}}{1 - 2\widetilde{e}_{ii}} \\ \Leftrightarrow\; & \underline{\underline{\widetilde{\lambda}_i = \frac{s_{ii}}{1 - r_{ii}}}}. \end{aligned}$$

Note: This is equivalent to the Rayleigh quotient $\dfrac{x_{(i)}^T A x_{(i)}}{x_{(i)}^T x_{(i)}}$.

Next we focus on the off-diagonal parts: From (*) and (**),

$$\begin{cases} \widetilde{e}_{ij} + \widetilde{e}_{ji} = r_{ij} \\ -\widetilde{\lambda}_i \widetilde{e}_{ij} - \widetilde{\lambda}_j \widetilde{e}_{ji} = s_{ij} \end{cases} \quad \text{for } i \neq j.$$

$\Rightarrow$ Here all $\widetilde{\lambda}_i$ have already been obtained at the previous step.

$\Rightarrow$ If $\widetilde{\lambda}_i \neq \widetilde{\lambda}_j$, then

$$\underline{\underline{\widetilde{e}_{ij} = \frac{s_{ij} + \widetilde{\lambda}_j r_{ij}}{\widetilde{\lambda}_j - \widetilde{\lambda}_i}}} \quad \text{for } i \neq j.$$

Otherwise (namely, $\widetilde{\lambda}_i = \widetilde{\lambda}_j$), for example

$$\underline{\underline{\widetilde{e}_{ij} = \frac{r_{ij}}{2}}} \quad \text{for } i \neq j.$$

Crude analysis

Assume $\|E\| = \varepsilon < 1$. It holds that

$$\begin{cases} E + E^T = R + \Delta_1, \quad \|\Delta_1\| = \mathcal{O}(\varepsilon^2) \\ D - DE - E^T D = S + \Delta_2, \quad \|\Delta_2\| = \mathcal{O}(\|A\|\varepsilon^2) \end{cases}.$$

Instead, we actually solve

$$\begin{cases} \widetilde{E} + \widetilde{E}^T = R \\ \widetilde{D} - \widetilde{D}\widetilde{E} - \widetilde{E}^T \widetilde{D} = S \end{cases}.$$

$$\Rightarrow \quad \|E - \widetilde{E}\| \sim \varepsilon^2$$

Then

$$\|X - X^{(k)}\| =: \varepsilon_k \quad \Rightarrow \quad \|X - X^{(k+1)}\| \sim \varepsilon_k^2$$

Algorithm 1: Refinement of approximate eigenvectors

Input: $A = A^{\mathrm{T}} \in \mathbb{R}^{n\times n}$; $\widehat{X} \in \mathbb{R}^{n\times \ell}$

Output: $X' \in \mathbb{R}^{n\times \ell}$; $\widetilde{D} = \mathrm{diag}(\widetilde{\lambda}_i) \in \mathbb{R}^{\ell\times\ell}$; $\widetilde{E} \in \mathbb{R}^{\ell\times\ell}$; $\delta \in \mathbb{R}$

function $[X', \widetilde{D}, \widetilde{E}, \delta] \leftarrow$ RefSyEv(A, $\widehat{X}$)

$\quad R \leftarrow I - \widehat{X}^{\mathrm{T}}\widehat{X}$

$\quad S \leftarrow \widehat{X}^{\mathrm{T}} A \widehat{X}$

$\quad \widetilde{\lambda}_i \leftarrow s_{ii}/(1 - r_{ii})$ for $i = 1, \cdots, \ell$ ▷ Approximate eigenvalues

$\quad \widetilde{D} \leftarrow \mathrm{diag}(\widetilde{\lambda}_i)$

$\quad \delta \leftarrow 2(\|S - \widetilde{D}\|_2 + \|A\|_2\|R\|_2)$

$$\widetilde{e}_{ij} \leftarrow \begin{cases} \dfrac{s_{ij} + \widetilde{\lambda}_j r_{ij}}{\widetilde{\lambda}_j - \widetilde{\lambda}_i} & \text{if } |\widetilde{\lambda}_i - \widetilde{\lambda}_j| > \delta \\ r_{ij}/2 & \text{otherwise} \end{cases} \quad \text{for } 1 \le i, j \le \ell$$

$\quad X' \leftarrow \widehat{X} + \widehat{X}\widetilde{E}$ ▷ Update $\widehat{X}$ by $\widehat{X}(I + \widetilde{E})$.

end function

Computational complexity: $\mathcal{O}(n^3)$ operations (for one iteration)

Convergence property

Theorem 1. *Let A be a real symmetric $n \times n$ matrix with the eigenvalues λ_i, $i = 1, 2, \ldots, n$. Suppose that Algorithm 1 is applied to A and $\widehat{X} \in \mathbb{R}^{n \times n}$, and $X' \in \mathbb{R}^{n \times n}$ is obtained. Define E and E' such that $X = \widehat{X}(I + E)$ and $X = X'(I + E')$, respectively. If*

$$\|E\| < \frac{\min_{i \neq j} |\lambda_i - \lambda_j|}{10n\|A\|}, \tag{5}$$

then

$$\|E'\| < \frac{5}{7}\|E\|, \tag{6}$$

$$\limsup_{\|E\| \to 0} \frac{\|E'\|}{\|E\|^2} \leq \frac{6n\|A\|}{\min_{i \neq j} |\lambda_i - \lambda_j|}. \tag{7}$$

Remark

- The algorithm can easily be extended to $Ax = \lambda Bx$ with B s.p.d., by setting $R := I - \widehat{X}^T B \widehat{X}$.
- The algorithm also works well for Hermitian matrices.
- Not working for a few eigenvectors but for all eigenvectors.

Numerical results (1)

- CPUs: 3.0 GHz Intel Xeon E5-2687W v4 (12 cores) $\times$ 2
- MATLAB R2016b with IEEE 754 binary64
- Advanpix Multiprecision Computing Toolbox (GMP and MPFR)

Table 7: Results for a pseudo-random real symmetric matrix, $n = 1000$

Algorithm 1	`eig` (bin64)	$\nu = 1$	$\nu = 2$	$\nu = 3$
$e_{\max}$	7.5×10^{-13}	2.8×10^{-25}	1.2×10^{-49}	2.1×10^{-98}
Time (sec.)	0.06	4.33	11.66	27.64
(accumulated)	0.06	4.38	16.04	43.68
MP-approach	`mp.Digits`	$\mathtt{d} = 34$	$\mathtt{d} = 54$	$\mathtt{d} = 103$
Time (sec.)		17.11	291.83	355.16

$e_{\max}$: (approximate) maximum error of eigenvectors

Future work

- Extension to SVD (already done!)
- Unsymmetric case

Thanks for your kind attention!