# Complete Decomposition of Symmetric Tensors in Linear Time and Polylogarithmic Precision

Subhayan Saha
(joint work with Pascal Koiran)

WACT 2023

LIP, ENS Lyon

# Outline

# Outline

# Symmetric Tensor Decomposition

$T \in \mathbb{C}^n \otimes \mathbb{C}^n \otimes \mathbb{C}^n$ - symmetric tensor, order-3

- Can be viewed as a 3-dimensional array $(T_{ijk})_{i,j,k \in [n]}$
- Invariant under permutations of indices
- 3-dimensional generalization of symmetric matrices

# Symmetric Tensor Decomposition

$T \in \mathbb{C}^n \otimes \mathbb{C}^n \otimes \mathbb{C}^n$ - symmetric tensor, order-3

- Can be viewed as a 3-dimensional array $(T_{ijk})_{i,j,k \in [n]}$
- Invariant under permutations of indices
- 3-dimensional generalization of symmetric matrices

Look at decompositions of the form:

$$T = \sum_{i=1}^{r} u_i \otimes u_i \otimes u_i \qquad (1)$$

where $u_i \in \mathbb{C}^n$.

- Smallest value of $r$ - symmetric tensor rank of $T$
- NP-hard to compute (Shitov,2016)

# Symmetric Tensor Decomposition

We still look at decompositions of the form:

$$T = \sum_{i=1}^{r} u_i \otimes u_i \otimes u_i$$

where $u_i \in \mathbb{C}^n$.

# Symmetric Tensor Decomposition

We still look at decompositions of the form:

$$T = \sum_{i=1}^{r} u_i \otimes u_i \otimes u_i$$

where $u_i \in \mathbb{C}^n$.

**Impose two additional conditions:**

1. $u_i$'s are linearly independent.
   - Decomposition unique (up to permutation and scaling by cube roots of unity), if it exists.
   - $r \leq n$ - undercomplete decompositions
2. $r = n$ - complete decompositions

# Symmetric Tensor Decomposition

We still look at decompositions of the form:

$$T = \sum_{i=1}^{r} u_i \otimes u_i \otimes u_i$$

where $u_i \in \mathbb{C}^n$.

**Impose two additional conditions:**

1. $u_i$'s are linearly independent.
   - Decomposition unique (up to permutation and scaling by cube roots of unity), if it exists.
   - $r \leq n$ - undercomplete decompositions
2. $r = n$ - complete decompositions

**Definition:** Tensor $T$ **diagonalisable** if it satisfies these conditions. Matrix $U$ - rows $u_1, \ldots, u_n$ **diagonalises** $T$

# Model of Computation

**Finite precision arithmetic:**

- Machine precision **u** - function of input size and desired accuracy.
- Input $x \in \mathbb{C}$ is stored as $\mathrm{fl}(x) = (1 + \Delta)x$ for some adversarially chosen $\Delta \in \mathbb{C}$ where $|\Delta| \leq u$
- Bit lengths of numbers stored - remain fixed at $\log(\frac{1}{u})$.

# Model of Computation

**Finite precision arithmetic:**

- Machine precision **u** - function of input size and desired accuracy.
- Input $x \in \mathbb{C}$ is stored as $\mathrm{fl}(x) = (1 + \Delta)x$ for some adversarially chosen $\Delta \in \mathbb{C}$ where $|\Delta| \leq u$
- Bit lengths of numbers stored - remain fixed at $\log(\frac{1}{u})$.
- Each arithmetic operation $* \in \{+, -, \times, \div\}$ is guaranteed to yield an output satisfying

$$\mathrm{fl}(x * y) = (x * y)(1 + \Delta) \text{ where } |\Delta| \leq u \tag{2}$$

# Algorithmic problem

**Approximate tensor decomposition:**
**Input:** Diagonalisable tensor $T = \sum_{i=1}^{n} u_i^{\otimes 3}$, $u_i$'s linearly independent, accuracy parameter $\epsilon$
**Goal:** Find linearly independent vectors $u'_1, \ldots, u'_n$ such that $u'_i$ are at $\leq \epsilon$-distance from $u_i$.

Forward approximation in the sense of numerical analysis - output solution close to the actual output.

# Outline

# Condition Number

Matrix $A \in \mathbb{C}^{m \times n}$ - $||A||_F = \sqrt{\sum_{i \in [m], j \in [n]} |A_{i,j}|^2}$ - Frobenius norm.

- $A$-invertible, $\kappa_F(A) = ||A||_F^2 + ||A^{-1}||_F^2$.
- Related to usual notion of condition number $\kappa(A) = ||A|| ||A^{-1}||$

# Condition Number

Matrix $A \in \mathbb{C}^{m \times n}$ - $||A||_F = \sqrt{\sum_{i \in [m], j \in [n]} |A_{i,j}|^2}$ - Frobenius norm.

- $A$-invertible, $\kappa_F(A) = ||A||_F^2 + ||A^{-1}||_F^2$.
- Related to usual notion of condition number $\kappa(A) = ||A||||A^{-1}||$

**Definition:** $T$ - diagonalisable tensor over $\mathbb{C}$, $U$ diagonalises $T$.
**Condition number** of $T$ $(\kappa(T)) = \kappa_F(U)$

**Lemma:** $T$-diagonalisable tensor. $\kappa(T)$ is well-defined (does not depend on choice of $U$).

# Results

**Input:** diagonalisable tensor $T$, desired accuracy parameter $\epsilon$ and estimate $B \geq \kappa(T)$

**Output:** $\epsilon$-approximate solution to the tensor decomposition problem for $T$

**Number of arithmetic operations:** $O(n^3 + T_{MM}(n)\log^2(\frac{nB}{\epsilon}))$

**Bits of precision:** poly-log$(n, B, \frac{1}{\epsilon})$

**Probability:** $1 - \frac{1}{8n}$

# Results

**Input:** diagonalisable tensor $T$, desired accuracy parameter $\epsilon$ and estimate $B \geq \kappa(T)$
**Output:** $\epsilon$-approximate solution to the tensor decomposition problem for $T$
**Number of arithmetic operations:** $O(n^3 + T_{MM}(n)\log^2(\frac{nB}{\epsilon}))$
**Bits of precision:** poly-log$(n, B, \frac{1}{\epsilon})$
**Probability:** $1 - \frac{1}{8n}$

Important conclusions:

- Bits of precision required = **polylogarithmic** in $n$, $B$ and $\frac{1}{\epsilon}$.
- Running time = $O(n^3)$ for all $\epsilon = \frac{1}{\text{poly(n)}}$, i.e., **linear** in the size of the input tensor (first such algorithm)
- Can provide inverse exponential accuracy, i.e., polynomial time even when $\epsilon = \frac{1}{\exp{(n)}}$.

# Related work

- Optimized version of Jennrich's algorithm/simultaneous diagonalisation.

# Related work

- Optimized version of Jennrich's algorithm/simultaneous diagonalisation.
- (Bhaskara et al, 2014)
  - algorithm runs in polynomial time in the exact arithmetic computation model (even when input has some noise)
  - Requires that the diagonalisation operation be done exactly

# Related work

- Optimized version of Jennrich's algorithm/simultaneous diagonalisation.
- (Bhaskara et al, 2014)
  - algorithm runs in polynomial time in the exact arithmetic computation model (even when input has some noise)
  - Requires that the diagonalisation operation be done exactly
- (Beltrán et al, 2019)
  - "pencil-based algorithms" for tensor decomposition are numerically unstable
  - We can escape this result because our algorithm is randomized.

# Outline

# Slices

Order-3 tensor $T \in \mathbb{C}^n \otimes \mathbb{C}^n \otimes \mathbb{C}^n$ can be "cut" into $n$ slices $T_1, \ldots, T_n \in M_n(\mathbb{K})$ where

$$(T_k)_{i,j} = (T_{ijk})_{1 \leq i,j \leq n}.$$

**Note:** For a symmetric tensor, each slice is a symmetric matrix of size $n$.

# Slices

Order-3 tensor $T \in \mathbb{C}^n \otimes \mathbb{C}^n \otimes \mathbb{C}^n$ can be "cut" into $n$ slices $T_1, \ldots, T_n \in M_n(\mathbb{K})$ where

$$(T_k)_{i,j} = (T_{ijk})_{1 \leq i,j \leq n}.$$

**Note:** For a symmetric tensor, each slice is a symmetric matrix of size $n$.

Let's look at some examples of slices:

If

$$T = \sum_{i=1}^{n} e_i^{\otimes 3},$$

then

$$(T_i)_{j,k} = 1 \text{ if } i = j = k \text{ and } 0 \text{ otherwise.}$$

# Jennrich's Algorithm (Symmetric version)

$T$-diagonalisable tensor, $T_1, \ldots, T_n$-slices of $T$

(i) Pick vectors $a = (a_1, \ldots, a_n)$ and $b = (b_1, \ldots, b_n)$ at random

(ii) Compute $T^{(a)} = \sum_{i=1}^{n} a_i T_i$ and $T^{(b)} = \sum_{i=1}^{n} b_i T_i$

(iii) Diagonalise $(T^{(a)})^{-1} T^{(b)} = VDV^{-1}$.

(iv) Let $w_1, \ldots, w_n$ be the rows of $V^{-1}$.

(v) Solve for $\alpha_i$ in $T = \sum_{i=1}^{n} \alpha_i w_i^{\otimes 3}$

(vi) Output $(\alpha_1)^{\frac{1}{3}} w_1, \ldots, (\alpha_n)^{\frac{1}{3}} w_n$.

# Why does it work?

Let $T = \sum_{i=1}^{n} u_i^{\otimes 3}$. $U$-rows $u_1, \ldots, u_n$

# Why does it work?

Let $T = \sum_{i=1}^{n} u_i^{\otimes 3}$. $U$-rows $u_1, \ldots, u_n$

- **Structure of slices:** $T_i = U^T \begin{pmatrix} u_{1i} & & \\ & \ddots & \\ & & u_{n,i} \end{pmatrix} U.$

# Why does it work?

Let $T = \sum_{i=1}^{n} u_i^{\otimes 3}$. $U$-rows $u_1, \ldots, u_n$

- **Structure of slices:** $T_i = U^T \begin{pmatrix} u_{1i} & & \\ & \ddots & \\ & & u_{n,i} \end{pmatrix} U.$
- Then

$$T^{(a)} = U^T \begin{pmatrix} \langle a, u_1 \rangle & & \\ & \ddots & \\ & & \langle a, u_n \rangle \end{pmatrix} U$$

# Why does it work?

Let $T = \sum_{i=1}^{n} u_i^{\otimes 3}$. $U$-rows $u_1, \ldots, u_n$

- **Structure of slices:** $T_i = U^T \begin{pmatrix} u_{1i} & & \\ & \ddots & \\ & & u_{n,i} \end{pmatrix} U.$
- Then
$$T^{(a)} = U^T \begin{pmatrix} \langle a, u_1 \rangle & & \\ & \ddots & \\ & & \langle a, u_n \rangle \end{pmatrix} U$$
- **Columns of** $U^{-1}$ **are eigenvectors of** $(T^{(a)})^{-1} T^{(b)}$.

  Eigenvalues of $(T^{(a)})^{-1} T^{(b)}$ distinct whp.


Contact: Subhayan Saha (subhayan.saha@ens-lyon.fr)

# Outline

# Looking at Step 5

**Step 3:** Diagonalisation algorithm on $(T^{(a)})^{-1}T^{(b)} = VMV^{-1}$
$V = U^{-1}\Lambda$, $\Lambda = \text{diag}(k_1, ..., k_n)$ - since eigenvalues distinct
**Need to find** scaling factors $k_i$ in Step 5.

# Looking at Step 5

**Step 3:** Diagonalisation algorithm on $(T^{(a)})^{-1}T^{(b)} = VMV^{-1}$
$V = U^{-1}\Lambda$, $\Lambda = \text{diag}(k_1, ..., k_n)$ - since eigenvalues distinct
**Need to find** scaling factors $k_i$ in Step 5.

- Usual idea: Solve a system of linear equations
- System has $n$ variables, $n^3$ equations - cannot achieve $O(n^3)$ even in exact arithmetic
- Need a numerically stable algorithm as well

# Looking at Step 5

**Step 3:** Diagonalisation algorithm on $(T^{(a)})^{-1}T^{(b)} = VMV^{-1}$
$V = U^{-1}\Lambda$, $\Lambda = \mathrm{diag}(k_1, ..., k_n)$ - since eigenvalues distinct
**Need to find** scaling factors $k_i$ in Step 5.

- Usual idea: Solve a system of linear equations
- System has $n$ variables, $n^3$ equations - cannot achieve $O(n^3)$ even in exact arithmetic
- Need a numerically stable algorithm as well

Our idea:

- Perform "change of basis" of $T$ by matrix $V$ , Compute the traces of the slices of new tensor
- Requires $O(n^3)$ arithmetic operations and is numerically stable.

# Change of basis

**Change of basis operation:** Apply map $A \otimes A \otimes A$ to a tensor $T$. ( $A \in M_n(\mathbb{C})$) - apply $A$ to each of the 3 components/modes of the input tensor.


Contact: Subhayan Saha (subhayan.saha@ens-lyon.fr)

# Change of basis

**Change of basis operation:** Apply map $A \otimes A \otimes A$ to a tensor $T$. ( $A \in M_n(\mathbb{C})$) - apply $A$ to each of the 3 components/modes of the input tensor.

- $T = \sum_{i=1}^{r} u_i^{\otimes 3} \implies (A \otimes A \otimes A).T = \sum_{i=1}^{r} (A^T u_i)^{\otimes 3}$.
- Via polynomial-tensor equivalence: Can be thought of as a change of variables, $g(x) = f(Ax)$.

# Change of basis

**Change of basis operation:** Apply map $A \otimes A \otimes A$ to a tensor $T$. ( $A \in M_n(\mathbb{C})$) - apply $A$ to each of the 3 components/modes of the input tensor.

- $T = \sum_{i=1}^{r} u_i^{\otimes 3} \implies (A \otimes A \otimes A).T = \sum_{i=1}^{r} (A^T u_i)^{\otimes 3}$.
- Via polynomial-tensor equivalence: Can be thought of as a change of variables, $g(x) = f(Ax)$.

$D = \sum_{i=1}^{n} e_i^{\otimes 3}$ - diagonal tensor. $T$ - diagonalisable tensor.
Then $T = (U \otimes U \otimes U).D$ for $U \in \mathrm{GL}_n(\mathbb{C})$

# Modified Algorithm

Replaced Step 5:
The algorithm proceeds as follows.

(i) Pick vectors $a = (a_1, ..., a_n)$ and $b = (b_1, ..., b_n)$ at random

(ii) Compute $T^{(a)} = \sum_{i=1}^{n} a_i T_i$ and $T^{(b)} = \sum_{i=1}^{n} b_i T_i$

(iii) Diagonalise $(T^{(a)})^{-1} T^{(b)} = VDV^{-1}$.

(iv) Let $w_1, ..., w_n$ be the rows of $V^{-1}$.

**(v) Let $T' = (V \otimes V \otimes V).T$. Let $T'_1, ..., T'_n$ be the slices of $T'$. Define $\alpha_i = \mathbf{Tr}(T'_i)$.**

(vi) Output $(\alpha_1)^{\frac{1}{3}} w_1, ..., (\alpha_n)^{\frac{1}{3}} w_n$.

Input tensor $T = \sum_{t=1}^{n} u_t^{\otimes 3}$. $U$ -rows $u_1, ..., u_n$.
Step (iii) outputs $V = U^{-1}\Lambda$ where $\Lambda = \text{diag}(k_1, ..., k_n)$, $k_i \neq 0$.
Recall that we want to find the scaling factors $k_i$.

Recall that for diagonal tensor $D$

$$U \text{ diagonalises } T \implies T = (U \otimes U \otimes U).D$$

Input tensor $T = \sum_{t=1}^{n} u_t^{\otimes 3}$. $U$ -rows $u_1, ..., u_n$.
Step (iii) outputs $V = U^{-1}\Lambda$ where $\Lambda = \text{diag}(k_1, ..., k_n)$, $k_i \neq 0$.
Recall that we want to find the scaling factors $k_i$.

Recall that for diagonal tensor $D$

$$U \text{ diagonalises } T \implies T = (U \otimes U \otimes U).D$$

$$T' = (U^{-1}\Lambda \otimes U^{-1}\Lambda \otimes U^{-1}\Lambda).T = (\Lambda \otimes \Lambda \otimes \Lambda).D$$

So $\text{Tr}(T'_i) = k_i^3$.

# Change of basis

**Algorithmic Problem:**
**Input:** $V \in M_n(\mathbb{C})$, symmetric tensor $T \in \mathbb{C}^n \otimes \mathbb{C}^n \otimes \mathbb{C}^n$
**Output:** $\mathrm{Tr}(S_1), \ldots, \mathrm{Tr}(S_n)$ where $S_1, \ldots, S_n$-slices of $S = (V \otimes V \otimes V).T$, We give an $O(n^3)$ algorithm for this problem.

# Idea:

Don't need to compute entire tensor after change of basis - too expensive

# Idea:

Don't need to compute entire tensor after change of basis - too expensive

**Lemma**

*Let* $S = (V \otimes V \otimes V).T$, $S_1, ..., S_n$*-slices of* $S$. *Then*

$$S_i = V^T D_i V \text{ where } D_i = \sum_{m=1}^{n} v_{m,i} T_m$$

# Idea:

Don't need to compute entire tensor after change of basis - too expensive

**Lemma**

*Let* $S = (V \otimes V \otimes V).T$, $S_1, \ldots, S_n$*-slices of* $S$. *Then*

$$S_i = V^T D_i V \text{ where } D_i = \sum_{m=1}^{n} v_{m,i} T_m$$

$$Tr(S_i) = Tr(V^T D_i V) = Tr(V^T V D_i) = Tr(V^T V(\sum_{m=1}^{n} v_{m,i} T_m))$$

$$= \sum_{m=1}^{n} v_{mi} Tr(V^T V T_m)$$

# Eigenvalue gaps

$A$ - diagonalisable matrix, $\lambda_1, ..., \lambda_n$-eigenvalues of $A$. Then

$$\text{gap}(A) := \min_{i \neq j} |\lambda_i - \lambda_j|$$

**Step 3:** Diagonalise $D := (T^{(a)})^{-1} T^{(b)}$

Use fast and numerically stable diagonalisation algorithm from [Banks et al'20]

Lower bounds on gap$(D)$ required for numerically stable diagonalisation.

$T = \sum_{i=1}^{n} u_i^{\otimes 3}$, $U \in M_n(\mathbb{C})$, rows $u_1, ..., u_n$, $T_1, .., T_n$-slices of $T$

Recall

$$T^{(a)} = U^T \begin{pmatrix} \langle a, u_1 \rangle & & \\ & \ddots & \\ & & \langle a, u_n \rangle \end{pmatrix} U$$

$$\mathrm{gap}(D) = \min_{i \neq j} \left| \frac{\langle b, u_i \rangle}{\langle a, u_i \rangle} - \frac{\langle b, u_j \rangle}{\langle a, u_j \rangle} \right| = \min_{i \neq j} \left| \frac{\langle b, u_i \rangle \langle a, u_j \rangle - \langle b, u_j \rangle \langle a, u_i \rangle}{\langle a, u_i \rangle \langle a, u_j \rangle} \right|$$

# Looking at polynomials

$$P^{kl}(\mathbf{x},\mathbf{y}) = \sum_{i,j\in[n]} p_{ij}^{kl} x_i y_j$$

where coefficients $p_{ij}^{kl} = u_{ik}u_{jl} - u_{il}u_{jk}$

$$|P^{kl}(a,b)| = |\langle b,u_i\rangle\langle a,u_j\rangle - \langle b,u_j\rangle\langle a,u_i\rangle|$$

**lower bds** for $P^{kl}(a,b)\ \forall k,l\in[n] \implies$ **lower bds** for $\text{gap}(A)$

# Probabilistic analysis

- Quadratic polynomial $P^{kl}$ emerges out of analysis for gap$(D)$
- Need to show that for random choices of $a, b$, $P^{kl}(a, b)$ is bounded far away from 0 with high probability.

# Probabilistic analysis

- Quadratic polynomial $P^{kl}$ emerges out of analysis for gap($D$)
- Need to show that for random choices of $a, b$, $P^{kl}(a, b)$ is bounded far away from 0 with high probability.

We follow a two-step process:

- First, we assume $a$ and $b$ are drawn from the uniform distribution on the hypercube $[-1, 1)^n$. Using Carbery-Wright inequalities, we can show this.
- Round the coordinates of $a$ and $b$ to obtain a point $(a', b')$ from the discrete grid. Use multivariate Markov inequality to show that the function value at $(a', b')$ is not too far.

# Probabilistic analysis

- Quadratic polynomial $P^{kl}$ emerges out of analysis for gap$(D)$
- Need to show that for random choices of $a, b$, $P^{kl}(a, b)$ is bounded far away from 0 with high probability.

We follow a two-step process:

- First, we assume $a$ and $b$ are drawn from the uniform distribution on the hypercube $[-1, 1)^n$. Using Carbery-Wright inequalities, we can show this.
- Round the coordinates of $a$ and $b$ to obtain a point $(a', b')$ from the discrete grid. Use multivariate Markov inequality to show that the function value at $(a', b')$ is not too far.

Inspired by construction of robust hitting sets from [Forbes,Shpilka, 2018]

# Future work

- Composition of numerically stable algorithms
- Undercomplete decompositions (number of summands $r < n$)
- Overcomplete decompositions (number of summands $r > n$)

# Thank You!